Complexidade

Prof. Marcus Vinícius Midena Ramos

Universidade Federal do Vale do São Francisco

8 de junho de 2021

marcus.ramos@univasf.edu.br
www.univasf.edu.br/~marcus.ramos

Bibliografia

Básica:

- *Introdução à Teoria da Computação (capítulo 7)*
  M. Sipser
  Thomson, 2006, 2ª edição

Complementar:

- *Foundations of Computer Science (capítulo 3)*
  A. V. Aho e J. D. Ullman
  W. H. Freeman & Company, 1995
- *Introduction to Automata Theory, Languages and Computation (capítulo 10)*
  J. E. Hopcroft, R. Motwani e J. D. Ullman
  Addison-Wesley, 2007, 3ª edição
- *Languages and Machines (capítulo 14 e 15)*
  T. A. Sudkamp
  Addison-Wesley, 2006, 3ª edição
- *Introduction to Languages and the Theory of Computation (capítulo 23)*
  J. C. Martin
  McGraw-Hill, 1991, 1ª edição

Roteiro

Decidibilidade e complexidade

- A decidibilidade não impõe limites nos recursos que são necessários para se resolver um problema;
- Recursos: tempo e espaço (memória);
- Certos problemas decidíveis podem demandar quantidades exageradas desses recursos, inviabilizando soluções que tenham interesse prático;
- A teoria da decidibilidade classifica os problemas em decidíveis (ou solucionáveis) e indecidíveis (ou não-solucionáveis);
- A teoria da complexidade classifica os problemas decidíveis em tratáveis e intratáveis.

Decidibilidade e complexidade

- O que são problemas tratáveis e intratáveis?
- Como identificar problemas nessas duas categorias?

Tempo de execução

Seja M uma MT determinística que pára sobre todas as entradas.

- O "tempo de execução", ou "complexidade de tempo", de M é a função $f : \mathbb{N} \to \mathbb{N}$, onde:
  - n é o comprimento da entrada w;
  - $f(n)$ é o número máximo de transições que M executa ao processar qualquer entrada de comprimento n.
- Diz-se que M roda em tempo $f(n)$;
- Diz-se que M é uma MT de tempo $f(n)$.

Tempo de execução

Seja M uma MT não-determinística que pára sobre todas as entradas.

- O "tempo de execução", ou "complexidade de tempo", de M é a função $f : \mathbb{N} \to \mathbb{N}$, onde:
  - n é o comprimento da entrada w;
  - $f(n)$ é o número máximo de transições que M executa em qualquer ramo da sua computação ao processar qualquer entrada de comprimento n.
- Diz-se que M roda em tempo $f(n)$;
- Diz-se que M é uma MT de tempo $f(n)$.

Notação O-grande

Definição

Sejam $f, g : \mathbb{N} \to \mathbb{R}^+$.

- Diz-se que $f(n) \in O(g(n))$, $f(n) = O(g(n))$ ou ainda $f(n)$ é $O(g(n))$, se existirem números inteiros e positivos c e n_0 tais que:

$$\forall n \geq n_0, f(n) \leq c \cdot g(n)$$

- Diz-se que $g(n)$ é um "limitante superior assintótico" para $f(n)$.
- Representa que f cresce mais lentamente do que g, a menos de fatores constantes suprimidos.

Exemplo

Seja $f(n) = 5n^3 + 2n^2 + 22n + 6$. Então, $f(n) = O(g(n))$ com $g(n) = n^3$, ou seja, $f(n) = O(n^3)$.
Prova:

- Considere $c = 6$;
- Considere $n_0 = 6$;
- Então, $5n^3 + 2n^2 + 22n + 6 \leq 6n^3, \forall n \geq n_0$.

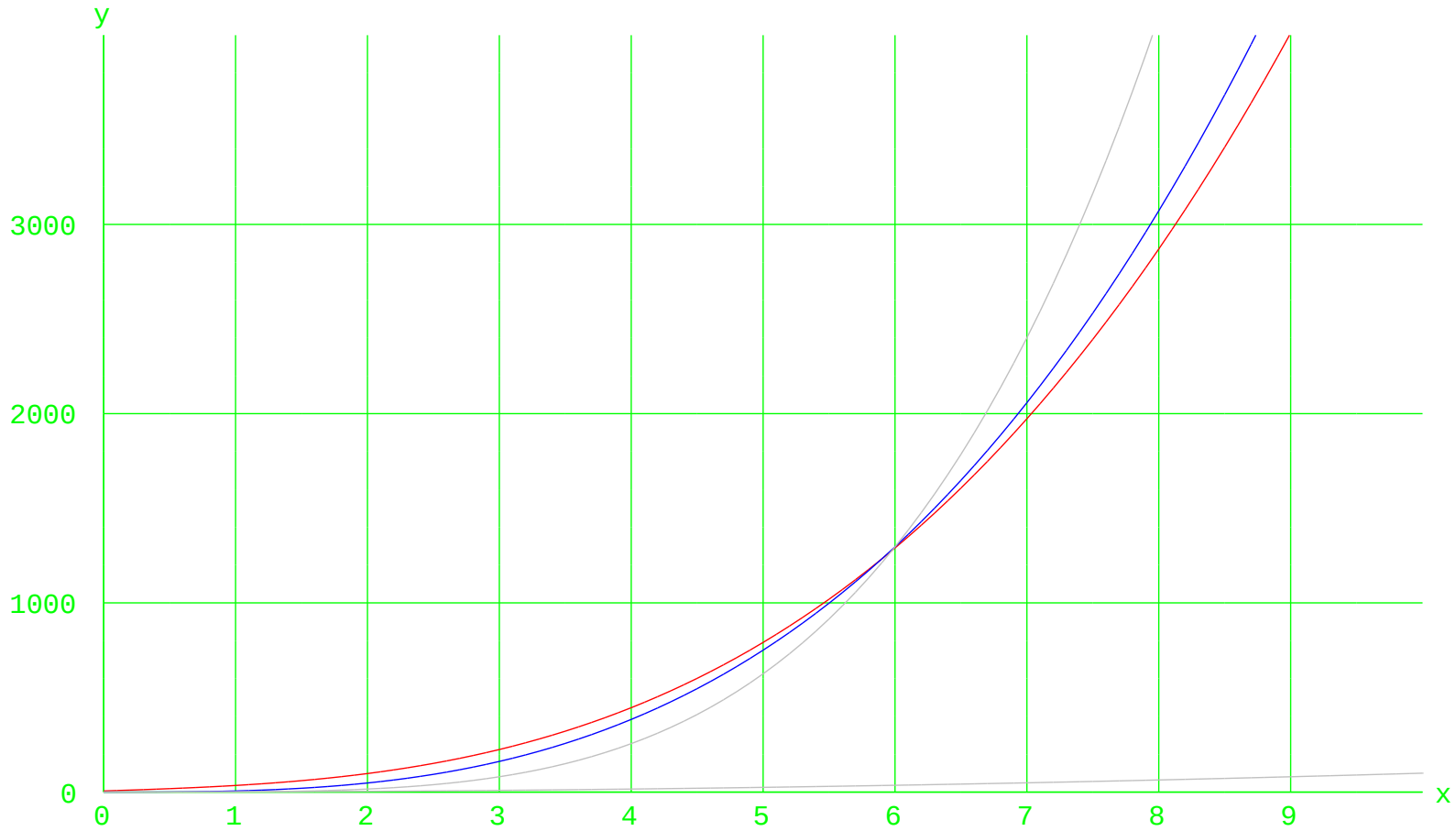
f também é $O(n^4)$, $O(n^5)$ etc.

Exemplo

No gráfico seguinte, as seguintes funções estão representadas:

- Em vermelho: $5n^3 + 2n^2 + 22n + 6$
- Em azul: $6n^3$
- Em cinza: n^4 e n^2

Exemplo



Exemplo

Seja $f(n) = n^2$. Então, $f(n)$ não é $O(n)$.
Prova:

- Supor que n^2 é $O(n)$;
- Então, devem existir n_0 e c tais que $\forall n \geq n_0, n^2 \leq c \cdot n$;
- Escolher $n_1 =$máx$(n_0, 2 \cdot c)$ (ou seja, $n_1 \geq n_0$ e $n_1 \geq 2 \cdot c$);
- Como $n_1 \geq n_0$, então $n_1^2 \leq c \cdot n_1$;
- Dividir os dois lados da inequação por n_1;
- Logo, $n_1 \leq c$;
- Mas isso é uma contradição, pois n_1 foi escolhido para satisfazer $n_1 \geq 0$ e $n_1 \geq 2 \cdot c$;
- A hipótese é falsa e n^2 não é $O(n)$.

Notação O-grande

Demonstrações

- Para provar que uma função $f(n)$ é $O(g(n))$, deve-se portanto:
  1. Escolher valores para n_0 e c;
  2. Provar que, $\forall n \geq n_0, f(n) \leq c \cdot g(n)$.
- Para provar que uma função $f(n)$ não é $O(g(n))$, deve-se:
  1. Provar que não existem n_0 e c tais que $\forall n \geq n_0, f(n) \leq c \cdot g(n)$.

Notação O-grande

Fatos

Princípios gerais observados:

1. Fatores constantes não importam:
   se $f(n) = c \cdot g(n)$, então $f(n)$ é $O(g(n))$ para qualquer $c > 0$.
2. Termos de ordem inferior não importam:
   se $f(n) = a_k n^k + a_{k-1} n^{k-1} + ... + a_2 n^2 + a_1 n + a_0$, com $a_k > 0$, então $f(n)$ é $O(a_k n^k)$ e também, pela regra anterior, $O(n^k)$.

Exemplos

Exemplos dos princípios:

1. Fatores constantes não importam:
   $2n^3$ é $O(0,001n^3)$: basta escolher $n_0 = 0$ e $c = 2000$.
2. Termos de ordem inferior não importam:
   $2^n + n^3$ é $O(2^n)$: basta escolher $n_0 = 10$ e $c = 2$.
   (observar que $\lim_{n\to\infty} \frac{n^3}{2^n} = 0$)

Notação O-grande

Propriedades

Se $f(n)$ é $O(g(n))$ e $\forall n, h(n) \geq 0$, então $h(n) \cdot f(n)$ é $O(h(n) \cdot g(n))$

- $\forall n \geq n_0$, então $f(n) \leq c \cdot g(n)$;
- A desigualdade é preservada ao se multiplicar ambos os termos por $h(n)$;
- $h(n) \cdot f(n) \leq h(n) \cdot c \cdot g(n)$, ou seja, $h(n) \cdot f(n) \leq c \cdot h(n) \cdot g(n)$;
- $h(n) \cdot f(n)$ é $O(h(n) \cdot g(n))$.

Notação O-grande

Propriedades

Se $\forall n \geq n_0, f(n) \leq g(n)$, então $f(n) + g(n)$ é $O(g(n))$

- Como $\forall n \geq n_0, f(n) \leq g(n)$, então $\forall n \geq n_0, f(n) + g(n) \leq g(n) + g(n)$;
- Portanto, $f(n) + g(n) \leq 2 \cdot g(n)$;
- $f(n) + g(n)$ é $O(g(n))$.

Notação O-grande

Propriedades

Se $f(n)$ é $O(g(n))$ e $g(n)$ é $O(h(n))$, então $f(n)$ é $O(h(n))$;

- $\forall n \geq n_1, f(n) \leq c_1 \cdot g(n)$;
- $\forall n \geq n_2, g(n) \leq c_2 \cdot h(n)$;
- Considerar $n_0 = \text{máx}(n_1, n_2)$;
- Considerar $c = c_1 \cdot c_2$;
- $\forall n \geq n_0$, então $n \geq n_1$ e $n \geq n_2$;
- Portanto, $f(n) \leq c_1 \cdot g(n)$ e $g(n) \leq c_2 \cdot h(n)$;
- Substituindo $g(n)$ por $c_2 \cdot h(n)$ em $f(n) \leq c_1 \cdot g(n)$, resulta $f(n) \leq c_1 \cdot c_2 \cdot h(n)$;
- $f(n)$ é $O(h(n))$.

Notação O-grande

Propriedades

Se $f_1(n)$ é $O(g(n))$, $f_2(n)$ é $O(h(n))$ e $g(n)$ é $O(h(n))$, então $f_1(n) + f_2(n)$ é $O(h(n))$;

- $\forall n \geq n_1, f_1(n) \leq c_1 \cdot g(n)$;
- $\forall n \geq n_2, f_2(n) \leq c_2 \cdot h(n)$;
- $\forall n \geq n_3, g(n) \leq c_3 \cdot h(n)$;
- Considerar $n_0 = \text{máx}(n_1, n_2, n_3)$;
- Portanto, $\forall n \geq n_0, f_1(n) + f_2(n) \leq c_1 \cdot g(n) + c_2 \cdot h(n)$;
- Substituindo $g(n)$ por $c_3 \cdot h(n)$, resulta $f_1(n) + f_2(n) \leq c_1 \cdot c_3 \cdot h(n) + c_2 \cdot h(n)$;
- $f_1(n) + f_2(n) \leq (c_1 \cdot c_3 + c_2) \cdot h(n)$;
- $f_1(n) + f_2(n)$ é $O(h(n))$.

Notação O-grande

Propriedades

Outras propriedades:

- Se $f(n)$ e $g(n)$ são polinômios, e o grau de $g(n)$ é maior ou igual que o grau de $f(n)$, então $f(n)$ é $O(g(n))$;
- Se $f(n)$ e $g(n)$ são polinômios, e o grau de $g(n)$ é menor que o grau de $f(n)$, então $f(n)$ não é $O(g(n))$;
- Se $f(n)$ é um polinômio, então $f(n)$ é $O(a^n), \forall a > 1$;
- $a^n, a > 1$ não é $O(f(n))$ para nenhum polinômio $f(n)$.

Notação O-grande

Funções

Procura-se, sempre que possível, fazer valer as seguinte regras básicas:

- Escolher a função $g(n)$ mais próxima de $f(n)$, tal que $f(n)$ seja $O(g(n))$;
- Escolher funções $g(n)$ que sejam *simples*:
  - Um único termo;
  - Coeficiente 1.

Notação O-grande

Funções

Funções simples mais utilizadas:

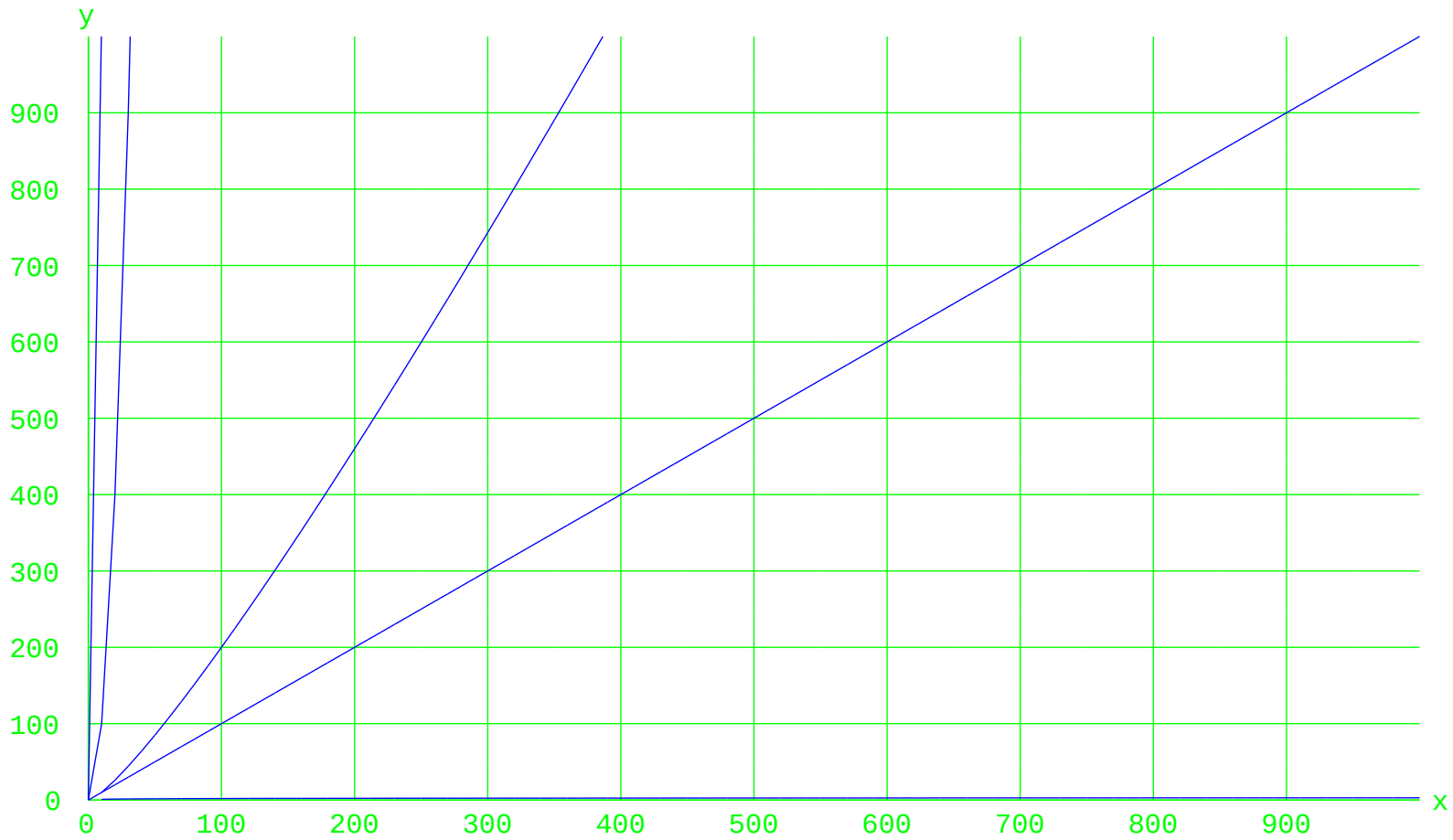
Big-Oh	Nome
$O(1)$	Constante
$O(\log n)$	Logarítmica
$O(n)$	Linear
$O(n \log n)$	$n \log n$
$O(n^2)$	Quadrática
$O(n^3)$	Cúbica
$O(2^n)$	Exponencial
$O(n!)$	Fatorial

Exemplo

No gráfico seguinte estão representadas as funções:

- $\log n$
- n
- $n \log n$
- n^2
- 2^n

Exemplo


y
900
800
700
600
500
400
300
200
100
0
0
100
200
300
400
500
600
700
800
900
x

Exemplo

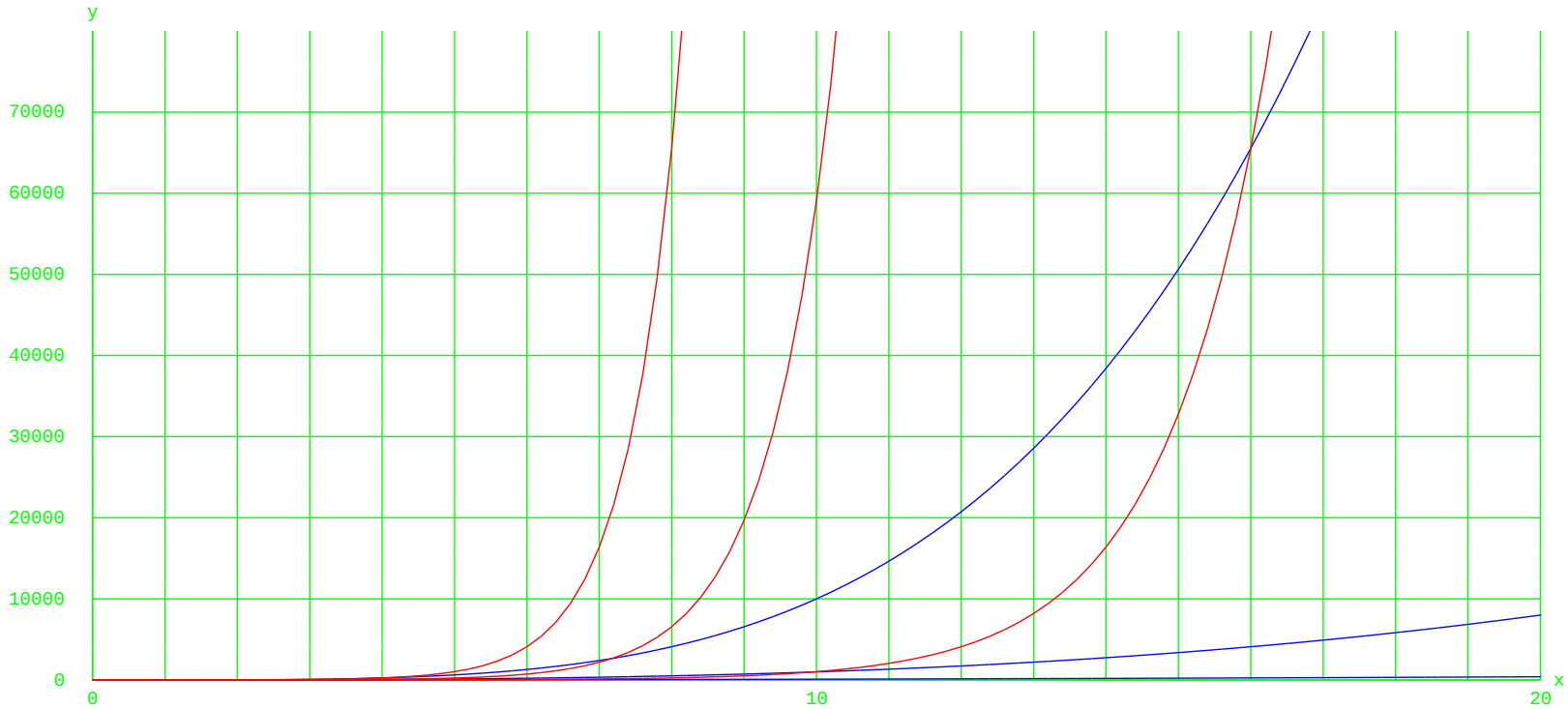
n	$\log_2(n)$	n	n^2	n^3	2^n	n!
5	2	5	25	125	32	120
10	3	10	100	1.000	1.024	3.628.800
20	4	20	400	8.000	1.048.576	$2{,}4*10^{18}$
30	5	30	900	27.000	$1{,}0*10^{9}$	$2{,}6*10^{32}$
40	5	40	1.600	64.000	$1{,}1*10^{12}$	$8{,}1*10^{47}$
50	6	50	2.500	125.000	$1{,}1*10^{15}$	$3{,}0*10^{64}$
100	7	100	10.000	1.000.000	$1{,}2*10^{30}$	$>10^{157}$
200	8	200	40.000	8.000.000	$1{,}6*10^{60}$	$>10^{374}$

Exemplo

No gráfico seguinte estão representadas as funções:

- n^2
- n^3
- n^4
- 2^n
- 3^n
- 4^n

Exemplo



Logaritmos

A mudança de base em logaritmos é feita pela multiplicação de um fator constante:

- Seja $f(n) = \log_2 n$
- Então, $\log_b n = \dfrac{\log_2 n}{\log_2 b}$
- Como $\log_2 b$ é um fator constante, então pode-se escrever $f(n) = O(\log(n))$ sem especificar a base, pois a notação O desconsidera fatores constantes por definição.

Exemplo:

- Seja $f(n) = 3n \log_2 n + 5n \log_2 \log_2 n + 2$
- Nesse caso, $f(n) = O(n \log n)$, pois $n \log n$ domina $n \log \log n$.

Notação O-pequeno

Sejam $f, g : \mathbb{N} \to \mathbb{R}^+$.

- Diz-se que $f(n) \in o(g(n))$, ou ainda $f(n) = o(g(n))$, se existirem números inteiros e positivos c e n_0 tais que:

$$\forall n \geq n_0, f(n) < c \cdot g(n)$$

- Diz-se que $f(n)$ é uma função "não mais" (ou "menor") que $g(n)$;
- Se $f(n) = o(g(n))$ então: $\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$.

Análise

Caso 1

Seja $A = \{0^k 1^k | k \geq 0\}$. Um possível algoritmo para decidir essa linguagem é representado pela MT M_1 descrita a seguir:

1. Varrer a fita inteira e rejeitar se houver algum símbolo 0 depois de um símbolo 1;
2. Repetir, enquanto existirem símbolos 0 e 1 na fita:
    3. Fazer uma varredura da fita, cortando um único 0 e um único 1;
4. Aceitar se todos os 0 e 1 tiverem sido cortados. Rejeitar se houver 0 ou 1 sobrando.

Análise

Caso 1

Cálculo do tempo de execução de M_1:

- Para a fase 1, são necessárias n transiçoes (ou "passos"). Para retornar a cabeça de leitura/escrita para a posição inicial, são necessários outros n passos. Logo, a fase 1 requer $2n$ passos, ou $O(n)$ passos;
- Para cada iteração das fases 2 e 3, são necessários $O(n)$ passos. Como cada iteração corta dois símbolos, serão necessárias $n/2$ varreduras da fita para cortar todos os símbolos. Logo, as fases 2 e 3 requerem, na sua totalidade, $\frac{n}{2} * O(n) = O(n^2)$ passos;
- A etapa 4 exige uma única varredura par decidir se aceita ou rejeita a entrada. Logo, ela requer $O(n)$ passos;
- O tempo total requerido por M_1 para analisar uma cadeia de comprimento n é, portanto, $O(n) + O(n^2) + O(n) = O(n^2)$.

Análise

Caso 2

Seja novamente $A = \{0^k 1^k | k \geq 0\}$. Um outro algoritmo para decidir essa linguagem é representado pela MT M_2 descrita a seguir:

1. Varrer a fita inteira e rejeitar se houver algum símbolo 0 depois de um símbolo 1;
2. Repetir, enquanto existirem símbolos 0 e 1 na fita:
    3. Fazer uma varredura na fita para determinar se a quantidade total de símbolos 0 e 1 é par ou ímpar; rejeitar se for ímpar;
    4. Fazer uma nova varredura, cortando 0s alternados e depois 1s alternados, começando do primeiro 0 e do primeiro 1;
5. Aceitar se a fita não contiver nenhum 0 e nenhum 1; rejeitar caso contrário.

Análise

Caso 2

Exemplo com $0^{13}1^{13}$ (aceita por M_2):

0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1
X	0	X	0	X	0	X	0	X	0	X	0	X	X	1	X	1	X	1	X	1	X	1	X	1	X
X	X	X	0	X	X	X	0	X	X	X	0	X	X	X	X	1	X	X	X	1	X	X	X	1	X
X	X	X	X	X	X	X	0	X	X	X	X	X	X	X	X	X	X	X	X	1	X	X	X	X	X
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

Análise

Caso 2

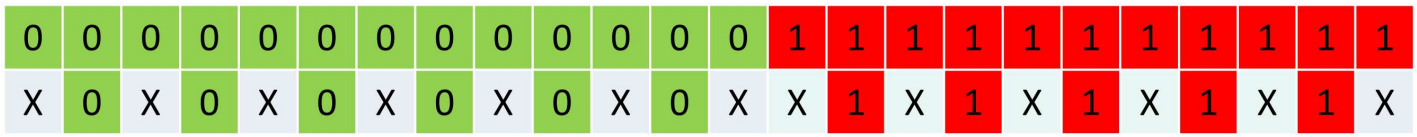
- A quantidade inicial de 0 é 13 (ímpar);
- Depois, ela é reduzida para 6 (par);
- Depois para 3 (ímpar);
- Finalmente para 1(ímpar);
- A mesma seqüência "ímpar, par, ímpar, ímpar" foi observada para a quantidade de símbolos 1;
- Se ímpar=1 e par=0, então as seqüências obtidas correspondem à 1011;
- Invertendo, obtemos 1101, que representa 13 em binário;
- Portanto, as quantidades de 0 e 1 são idênticas e a cadeia pertence à linguagem.

Análise

Caso 2

Exemplo com $0^{13}1^{12}$ (rejeitada por M_2 pois $13+12=25$, que não é par):

Exemplo com $0^{13}1^{11}$ (rejeitada por M_2 pois $6+5=1$, que não é par):



Análise

Caso 2

Seja $w = 0^m 1^n$:

- A representação binária de m (ou n) é feita por divisões inteiras sucessivas por 2; os restos representam os algarismos dessa representação, do menos para o mais significativo;
- Suponha que $m + n$ é par; então, $m = 2 * i + r, n = 2 * j + s$, com $0 \leq r, s \leq 1$, e $2 * i + r + 2 * j + s$ é par, ou seja, $r = s = 0$ ou $r = s = 1$;
- Logo, $m + n$ é par $\Rightarrow$ o algarismo menos significativo na representação binária de m é igual ao algarismo menos significativo na representação binária de n;
- A marcação alternada representa uma divisão por 2 inteira de m e n (os restos r e s são desprezados);

Análise

Caso 2

- A iteração dos passos anteriores (verificação da paridade e marcação alternada) garante que os algarismos seguintes na representação binária de m e n são idênticos também;
- Se não restar nenhum símbolo 0 ou 1, então a quantidade de dígitos que representam m e n são idênticas e, além disso, os dígitos de mesma posição em ambas as representações são idênticos também;
- Portanto, os números binários que representam m e n são idênticos, $m = n$ e $w \in A$.

Análise

Caso 2

Em resumo:

$$w = 0^m 1^n$$

- $m_2 = x_i x_{i-1} ... x_1 x_0$;
- $n_2 = y_j y_{j-1} ... y_1 y_0$;
- $(m = n) \Leftrightarrow (i = j) \wedge (x_k = y_k, 0 \leq k \leq i)$.

Análise

Caso 2

Cálculo do tempo de execução de M_2:

- Cada uma das etapas 1,3,4 e 5 demanda uma varredura na fita que é $O(n)$;
- As etapas 1 e 5 são executadas uma única vez, e portanto são $O(n)$;
- A etapa 4 corta à metade a quantidade de 0s e 1s cada vez que ela é executada;
- Portanto, o loop é executado no máximo $1 + \log_2 n$ vezes (adiciona-se 1 para poder zerar a quantidade de símbolos na fita; por exemplo, $\log_2 16 = 4$ mas são necessárias 5 iterações para chegar sucessivamente em 8, 4, 2, 1 e 0 símbolos na fita);

Análise

Caso 2

Cálculo do tempo de execução de M_2 (continuação):

- Logo, o tempo total de execução das etapas 2, 3 e 4 é $(1 + \log_2 n) * (O(n) + O(n))$, ou seja, $O(n \log n)$;
- O tempo total de execução de M_2 é $O(n) + O(n \log n) + O(n) = O(n \log n)$.

Análise

Caso 3

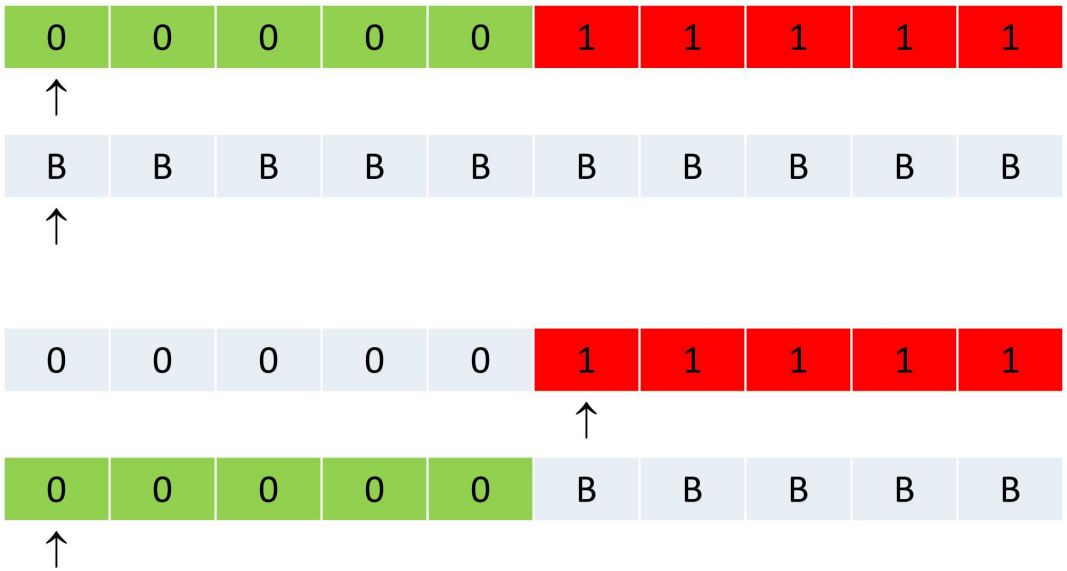
Seja novamente $A = \{0^k 1^k | k \geq 0\}$. Um outro algoritmo para decidir essa linguagem é representado pela MT M_3, dessa vez com duas fitas, descrita a seguir:

1. Varrer a primeira fita inteira e rejeitar se houver algum símbolo 0 depois de um símbolo 1;
2. Varrer novamente a primeira fita até encontrar o primeiro 1, porém copiando os símbolos 0 encontrados para a segunda fita;
3. Varrer os símbolos 1 da primeira fita, cortando um 1 dessa fita para cada 0 encontrado na segunda fita;
4. Se restarem 0 na segunda fita ou 1 na primeira fita, rejeitar; caso contrário, aceitar.

Análise

Caso 3

Exemplo com $0^5 1^5$ (aceita por M_3):



Análise

Caso 3

Cálculo do tempo de execução de M_3:

- Cada uma das etapas 1,2,3 e 4 demanda uma varredura na fita que é $O(n)$;
- Cada uma das etapas 1,2,3 e 4 é executada uma única vez;
- Portanto, o tempo total de execução de M_3 é $O(n)$.

Análise

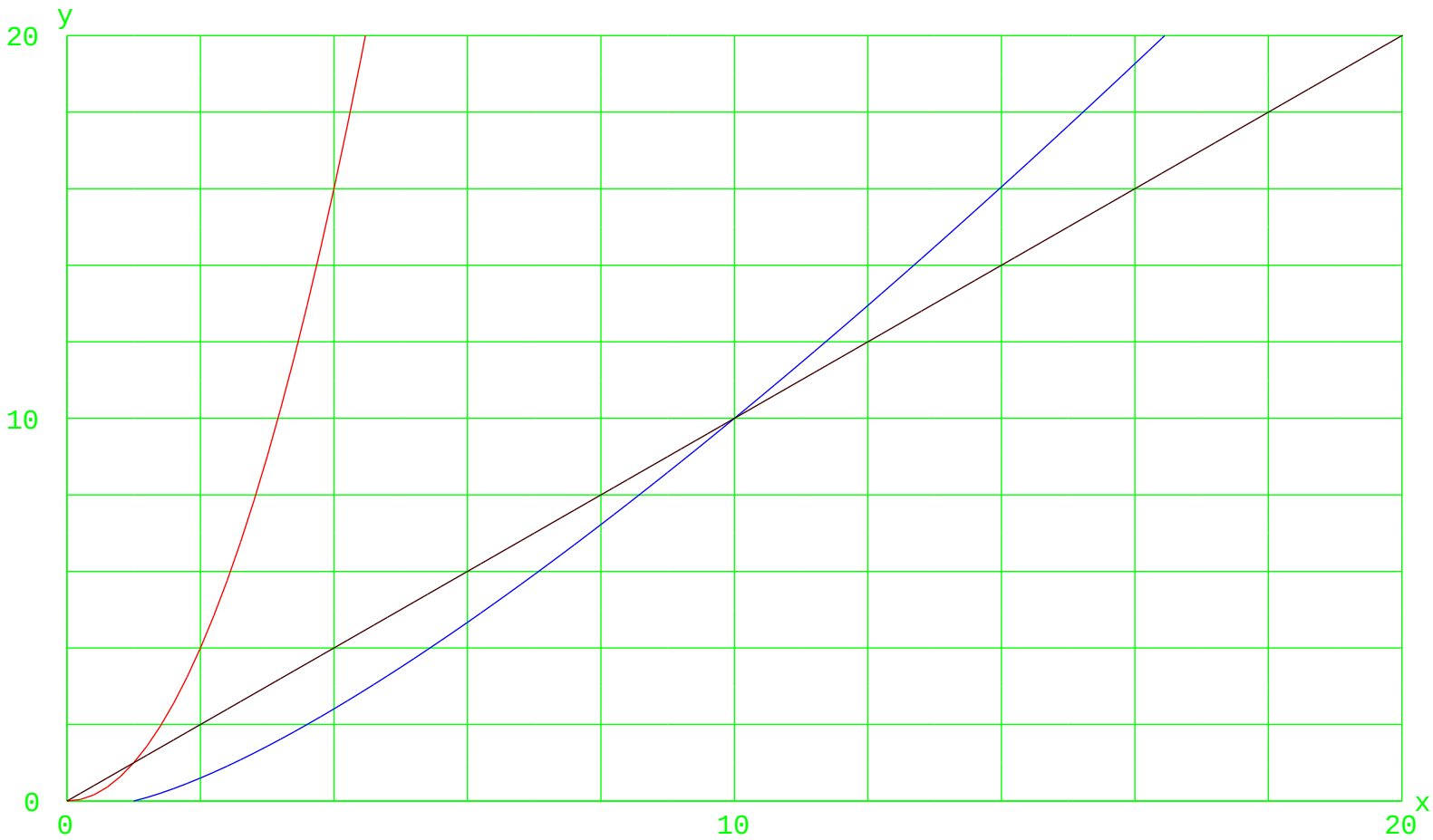
Resumo

Para a linguagem $A = \{0^k 1^k | k \geq 0\}$:

- M_1 decide A em $O(n^2)$;
- M_2 decide A em $O(n \log n)$;
- M_3 decide A em $O(n)$;
- A complexidade de tempo de A depende do modelo de computação escolhido.

Observação: a simulação de M_3 é feita por uma máquina de uma única fita de tempo $O(n^2)$.

Exemplo



Análise

Resumo

Decidibilidade x Complexidade:

- Na teoria da decidibilidade, a escolha do modelo de computação não muda as conclusões sobre a decibibilidade de uma certa linguagem;
- Na teoria da complexidade, a escolha do modelo de computação pode resultar em complexidades de tempo diferentes para uma mesma linguagem;
- Mas se a teoria da complexidade procura exatamente classificar os problemas em função da sua complexidade de tempo, qual modelo de computação deve ser usado?
- Os resultados apresentados pelos modelos determinísticos não sofrem grandes variações;
- A idéia é criar um sistema de classificação que não seja sensível à pequenas variações de complexidade.

Modelos e complexidades de tempo

Relembrando alguns resultados anteriormente vistos:

1. Toda MT de múltiplas fitas e tempo $t(n) \geq n$ possui uma MT equivalente de uma única fita com tempo $O(t(n)^2)$.
   A eliminação das fitas adicionais transforma o tempo de execução por no máximo uma potência de 2 do tempo original.
2. Toda MT não-determinística de uma única fita e tempo $t(n) \geq n$ possui uma MT equivalente determinística de uma única fita com tempo $O(2^{t(n)})$.
   A eliminação de não-determinismos transforma o tempo de execução por no máximo uma exponencial do tempo original.

As classes $TIME, DTIME$ e $NTIME$

- $TIME(t(n)) =$
  $\{L \mid L$ é uma linguagem decidida por uma MT
  de tempo $O(t(n)\}$
- $DTIME(t(n)) =$
  $\{L \mid L$ é uma linguagem decidida por uma MT determinística
  de tempo $O(t(n)\}$
- $NTIME(t(n)) =$
  $\{L \mid L$ é uma linguagem decidida por uma MT não-determinística
  de tempo $O(t(n)\}$

Exemplo

- $M_1 : A \in TIME(n^2)$;
- $M_1 : A \in DTIME(n^2)$;
- $M_1 : A \in NTIME(n^2)$;
- $M_2 : A \in TIME(n \log n)$;
- $M_2 : A \in DTIME(n \log n)$;
- $M_2 : A \in NTIME(n \log n)$;
- $M_3 : A \in TIME(n)$;
- $M_3 : A \in DTIME(n^2)$;
- $M_3 : A \in NTIME(n^2)$.

Introdução

- Tempos de execução polinomiais são considerados razoáveis, e as diferenças entre eles são consideradas pequenas;
- Tempos de execução exponenciais são considerados não razoáveis, e as diferenças entre eles e os tempos polinomiais são consideradas grandes;
- Seja $n = 1.000$:
  - $1.000^3 = 1.000.000.000$, grande porém administrável;
  - $2^{1.000}$ é muito maior que o número de átomos no universo observável (estimado em $\sim 10^{80}$);
- Algoritmos de tempo polinomial geralmente são suficientemente rápidos (considerando-se os modelos e a tecnologia da computação atual);
- Algoritmos de tempo exponencial raramente são úteis (idem);

Introdução

- Algoritmos de tempo exponencial geralmente surgem quando há busca exaustiva num espaço de possíveis soluções;
- Algoritmos de tempo polinomial surgem quando há um entendimento mais profundo sobre a natureza do problema;
- Como os vários modelos determinísticos de computação são todos polinomialmente equivalentes, diferenças dessa ordem serão ignoradas;
- Serão consideradas apenas as diferenças entre tempos polinomiais e exponenciais, estes últimos produzidos por modelos não-determinísticos;
- Essa delimitação facilita o estudo da teoria, mas na prática todas as diferenças de tempo são relevantes e devem ser consideradas.

Definição

$\mathcal{P}$ é a classe das linguagens que podem ser decididas em tempo polinomial por uma MT determinística:

$$\mathcal{P} = \bigcup_k DTIME(n^k)$$

Propriedades

A classe $\mathcal{P}$:

- É "matematicamente robusta", ou seja, ela não depende no particular modelo de computação que esteja sendo usado, desde que esse seja polinomialmente equivalente à Máquina de Turing determinística;
- Corresponde aproximadamente à classe de problemas que são possíveis de serem solucionados em um computador moderno, ou seja, ela apresenta relevância prática;

Características

Sempre que um algoritmo de tempo polinomial é descoberto para um algoritmo que até então parecia requerer apenas tempo exponencial, isso significa, geralmente, que:

- Foi obtido algum entendimento mais profundo sobre a natureza do problema e sobre a forma de se conseguir uma solução para o mesmo;
- Novas reduções de tempo podem ser obtidas na medida em que esse entendimento for aprofundado, até finalmente alcançar uma implementação suficientemente rápida para ser executada em computadores modernos.

Exemplos de problemas em $\mathcal{P}$

- $CAM =$
  $\{\langle G, s, t\rangle \mid G$ é um grafo direcionado que tem um caminho de s para $t\}$
- $PRIM_{ES} =$
  $\{\langle x, y\rangle \mid x$ e y são primos entre si$\}$[1]
- $LLC =$
  $\{\langle L, w\rangle \mid L$ é uma linguagem livre de contexto e $w \in L\}$

[1] Números inteiros primos entre si são aqueles que possuem como único divisor comum o número 1.

Problema CAM

$$CAM = \{\langle G, s, t\rangle \mid G \text{ é um grafo direcionado que tem um caminho de } s \text{ para } t\}$$

Teorema: $CAM \in \mathcal{P}$.

Prova:

1. Marcar o nó s;
2. Repetir até que nenhum novo nó tenha sido marcado:
   3. Verificar todas as arestas de G e marcar todos os nós b tais que (a, b) seja uma aresta e a seja um nó já marcado;
4. Se t estiver marcado, então aceitar; senão, rejeitar.

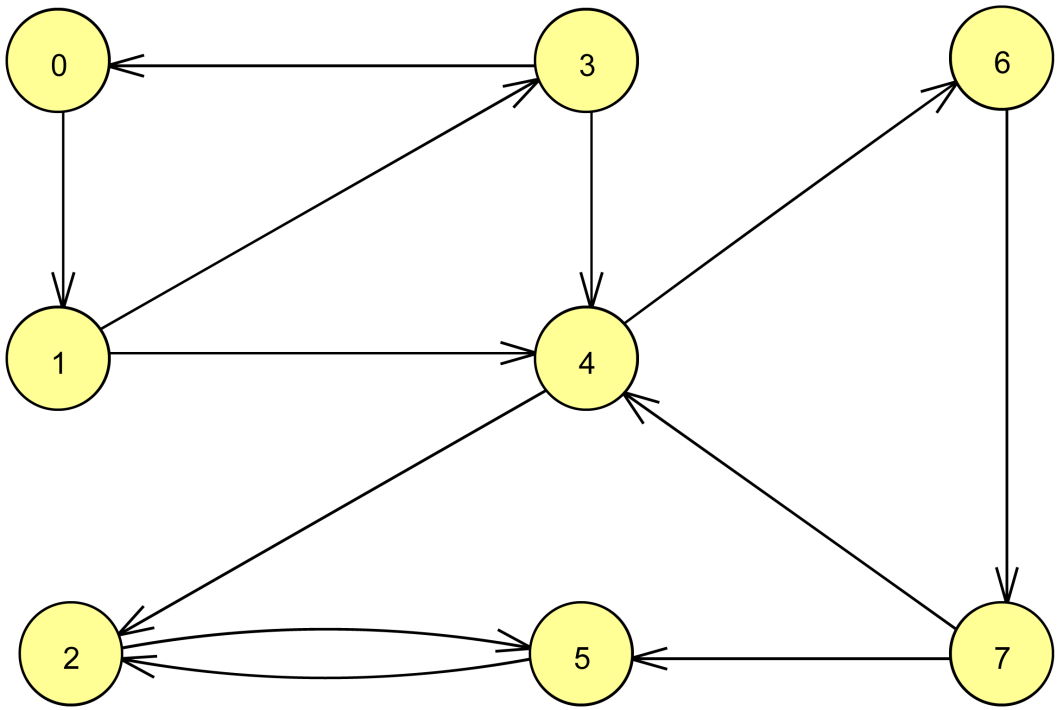
Problema CAM

Cálculo do tempo de execução (suponha que G tem n nós):

- A etapa 1 é proporcional à n, portanto ela é $O(n)$;
- A etapa 2 é repetida n vezes, uma para cada nó de G;
- A etapa 3 requer analisar se existem arestas com outros $(n-1)$ nós;
- As etapas 2 e 3 combinadas requerem $n * (n-1)$ análises, portanto ela é $O(n^2)$;
- A etapa 4 é proporcional à n, portanto ela é $O(n)$;
- O tempo total é $O(n) + O(n^2) + O(n) = O(n^2)$;
- $CAM \in (P)$.

Problema CAM

Exemplo



Problema CAM

Exemplo

Determinar se há caminho entre os nós 0 e 5:

- $\{0\},\ 0 : \{(0,1)\}$
- $\{0,1\},\ 1 : \{(1,3),(1,4)\}$
- $\{0,1,3,4\},\ 3 : \{(3,0),(3,4)\}$ e $4 : \{(4,2),(4,6)\}$
- $\{0,1,2,3,4,6\},\ 2 : \{(2,5)\}$ e $6 : \{(6,7)\}$
- $\{0,1,2,3,4,5,6,7\},\ 5 : \{(5,2)\}$ e $7 : \{(7,4),(7,5)\}$
- $\{0,1,2,3,4,5,6,7\}$
- Como $5 \in \{0,1,2,3,4,5,6,7\}$, então existe um caminho entre os nós 0 e 5;
- De fato, existem vários caminhos: 01425, 013425, 014675, 0134675 etc.

Problema CAM

Exemplo

Determinar se há caminho entre os nós 2 e 6:

- $\{2\},\ 2 : \{(2,5)\}$
- $\{2,5\},\ 5 : \{(5,2)\}$
- $\{2,5\}$
- Como $6 \notin \{2,5\}$, então não existe um caminho entre os nós 2 e 6.

Introdução

- A pesquisa exaustiva por uma solução dentro de um espaço de possíveis soluções (conhecido como método da "força bruta") geramente produz algoritmos de tempo exponencial;
- Todo problema decidível pode ser resolvido pelo método da força bruta;
- Diversos problemas de interesse prático possuem apenas algoritmos de tempo exponencial;
- Inúmeras tentativas tem sido feitas ao longo dos anos para encontrar algoritmos de tempo polinomial para esses problemas, porém sem sucesso até o momento;
- O motivo desse insucesso é desconhecido:
  - Talvez os respectivos algoritmos de tempo polinomiais ainda não tenham sido descobertos;
  - Talvez eles nem existam, ou seja, talvez os correspondentes problemas sejam intrinsicamente difíceis.

Definição

$\mathcal{NP}$ é a classe das linguagens que podem ser decididas em tempo polinomial por uma MT não-determinística:

$$\mathcal{NP} = \bigcup_k NTIME(n^k)$$

Propriedades

A classe $\mathcal{NP}$:

- É insensível ao modelo de computação não-determinístico selecionado, uma vez que eles são todos polinomialmente equivalentes;
- Contém muitos problemas de interesse prático.

Exemplos de problemas em $\mathcal{NP}$

- Caminho Hamiltoniano ($CAMHAM$);
- $CAMHAM = \{\langle G, s, t\rangle \mid G$ é um grafo direcionado que tem um caminho de s para t que passa por todos os demais nós de G exatamente uma vez$\}$
- Pode ser resolvido por uma MT não-determinística de tempo polinomial, ou seja por um MT determinística de tempo exponencial.
- Não se sabe se existem MT determinísticas de tempo polinomial que resolvam o mesmo.

Problema $CAMHAM$

MT não-determinística que decide $CAMHAM$ em tempo polinomial:

1. Gerar uma lista de m números $p_1, p_2, ..., p_m$, onde m é o número de nós de G. Os números p_i são escolhidos de forma não-determinística;
2. Se houver repetições de números na lista, rejeitar;
3. Se $p_1 \neq s$ ou $p_m \neq t$, rejeitar;
4. $\forall i, 1 \leq i \leq m-1$, verificar se (p_i, p_{i+1}) é uma aresta de G; se alguma não for, rejeitar, caso contrário, aceitar.

Como todas as etapas são executadas em tempo polinomial, o algoritmo executa em tempo polinomial (numa MT não-determinística).

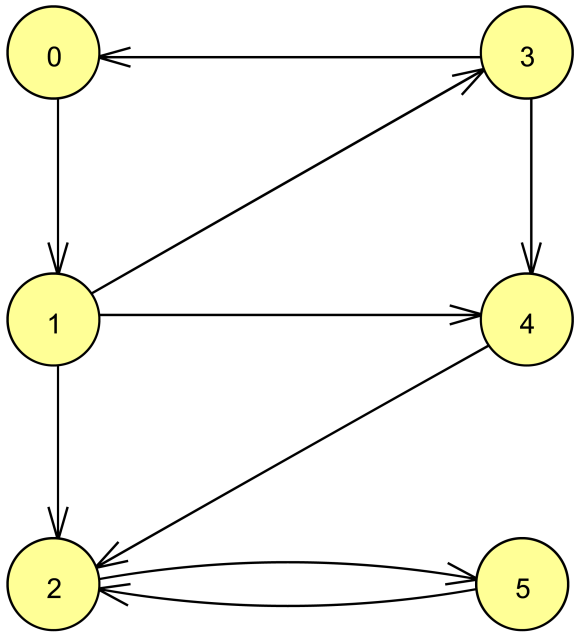
Problema $CAMHAM$

Exemplo

- Suponha G com 6 nós, denotados 0 a 5. A escolha não-determinística de uma cadeia com 6 nós pode ser feita, por exemplo, pela Máquina de Turing M, representada parcialmente a seguir;
- Em cada ramo da sua computação, M produz uma cadeia (possível solução do problema) diferente e grava a mesma numa fita inicialmente preenchida apenas com brancos;
- A verificação da validade de cada cadeia como solução do problema é feita partir do estado q_{10};
- M possui 6^6 seqüências distintas de movimentação, uma para cada cadeia possível de ser gerada de forma não-determinística (000000 até 555555).

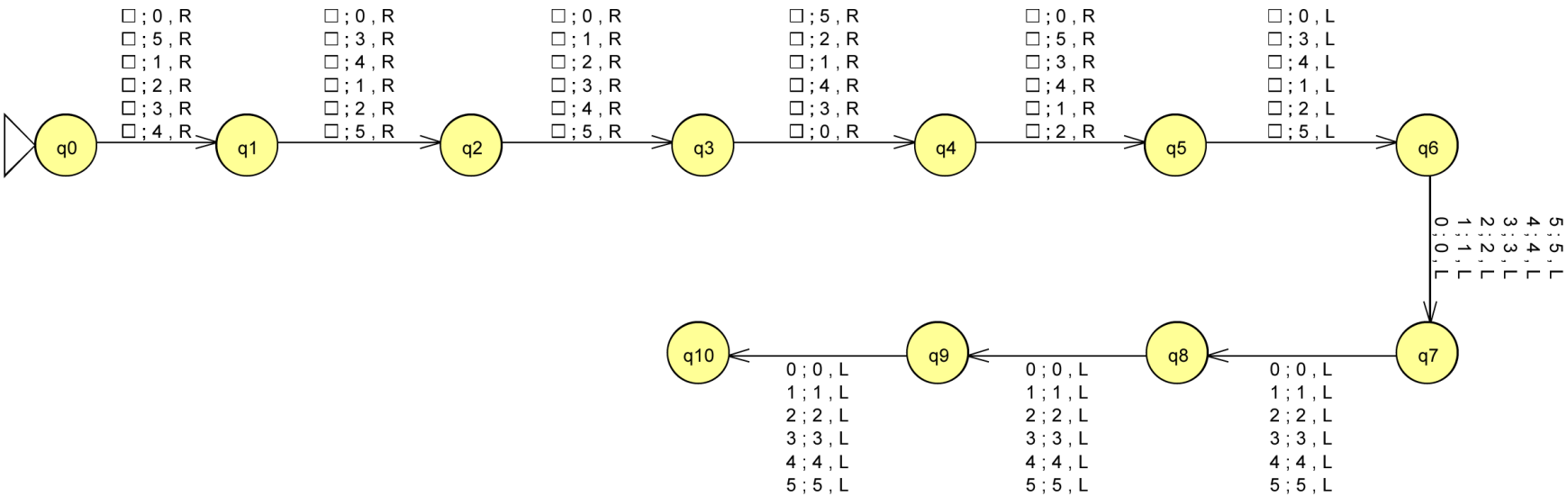
Problema $CAMHAM$

Exemplo



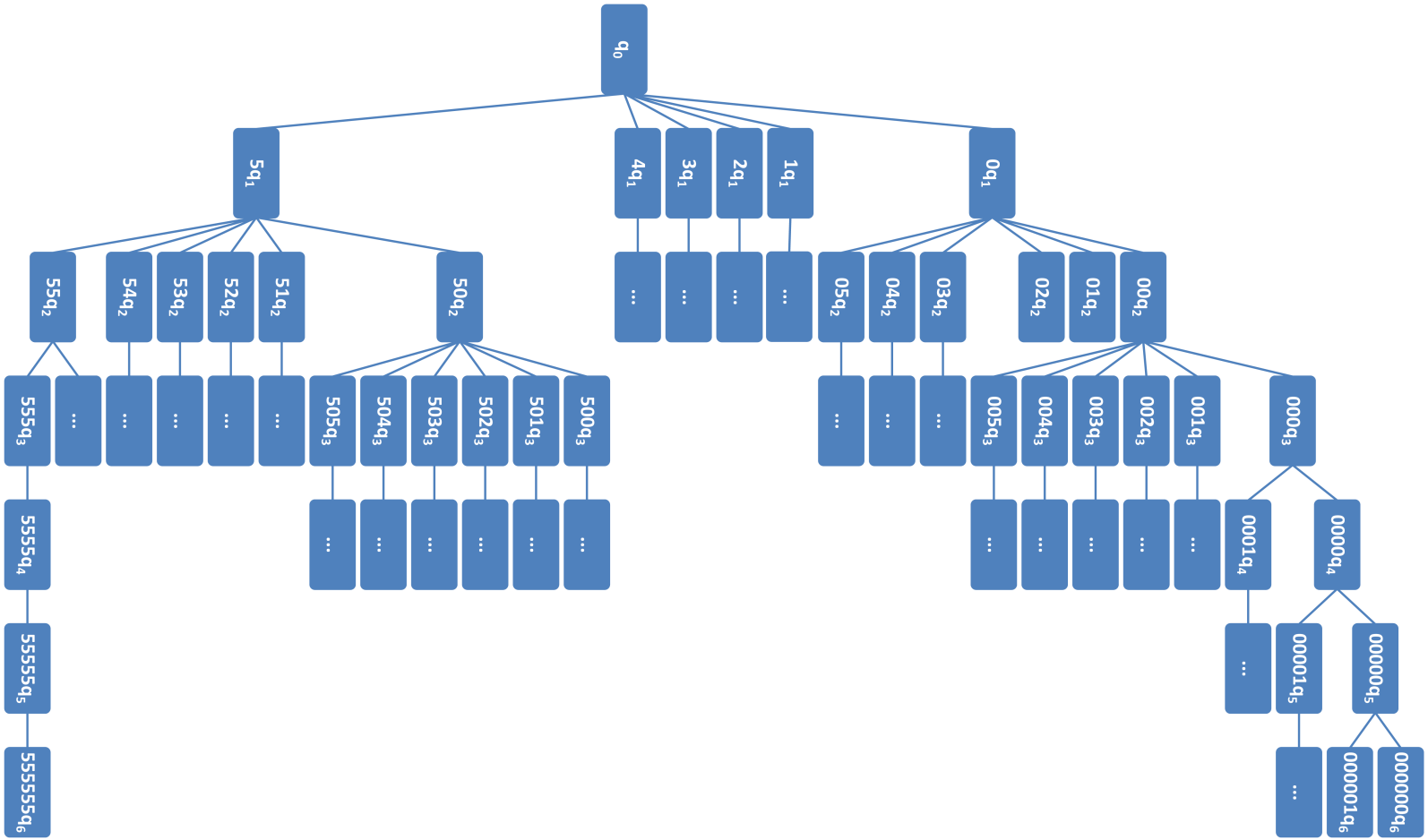
Problema $CAMHAM$

Exemplo



Problema $CAMHAM$

Exemplo



Problema $CAMHAM$

Exemplo

Suponha $s = 0$ e $t = 5$. Cadeias possíveis geradas pela Máquina de Turing não-determinística:

- 420531: rejeita, pois $s = 4$ (deveria ser 0);
- 025143: rejeita, pois $t = 3$ (deveria ser 5);
- 013245: rejeita, pois 2 não é sucessor de 3;
- 135502: rejeita, pois há repetição do nó 5;
- 013425: aceita.

Exemplos de problemas em $\mathcal{NP}$

$$COMPOSTOS = \{x \mid x = p \cdot q, \text{ para inteiros } p, q > 1\}$$

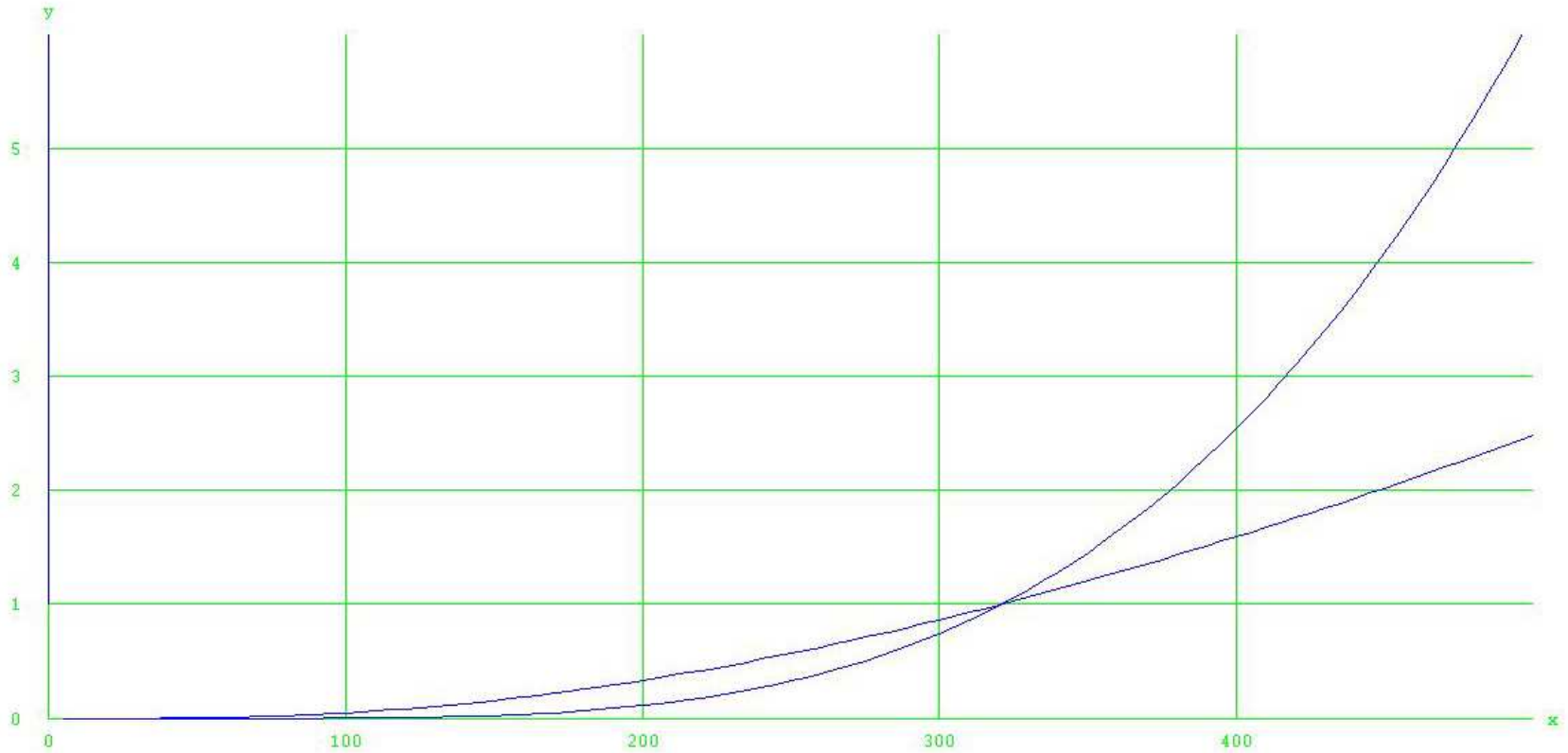
- Pode ser resolvido por uma MT não-determinística de tempo polinomial, ou seja por uma MT determinística de tempo exponencial;
- O tamanho da entrada n é medido pelo número de algarismos do número que se deseja testar $(n = log_{10}(x))$;
- Os algoritmos mais simples possuem tempo exponencial em n (como por exemplo gerar todos os números com n algarismos e testar um por um para verificar se x é divisível por ele, analisando depois o resto).

Problema $COMPOSTOS$

- Até recentemente suspeitava-se que $COMPOSTOS \in \mathcal{P}$ mas não havia prova disso;
- Em 2002 foi descoberto um algoritmo de tempo polinomial para esse problema, ou seja, foi provado que $COMPOSTOS \in \mathcal{P}$.
- Teste de primalidade de AKS (Agrawal-Kayal-Saxena, Indian Institute of Technology) com tempo $\tilde{O}(log^{12}(x))$;
- Investigações posteriores conseguiram reduzir esse tempo para $\tilde{O}(log^{6}(x))$;
- $\tilde{O}(g(n)) = O(g(n) * log^{k}(g(n)))$ para algum valor de k.

Problema *COMPOSTOS*

Curvas para $10^x, log^{12}(x) * log(log^{12}(x))$ e $log^6(x) * log(log^6(x))$:



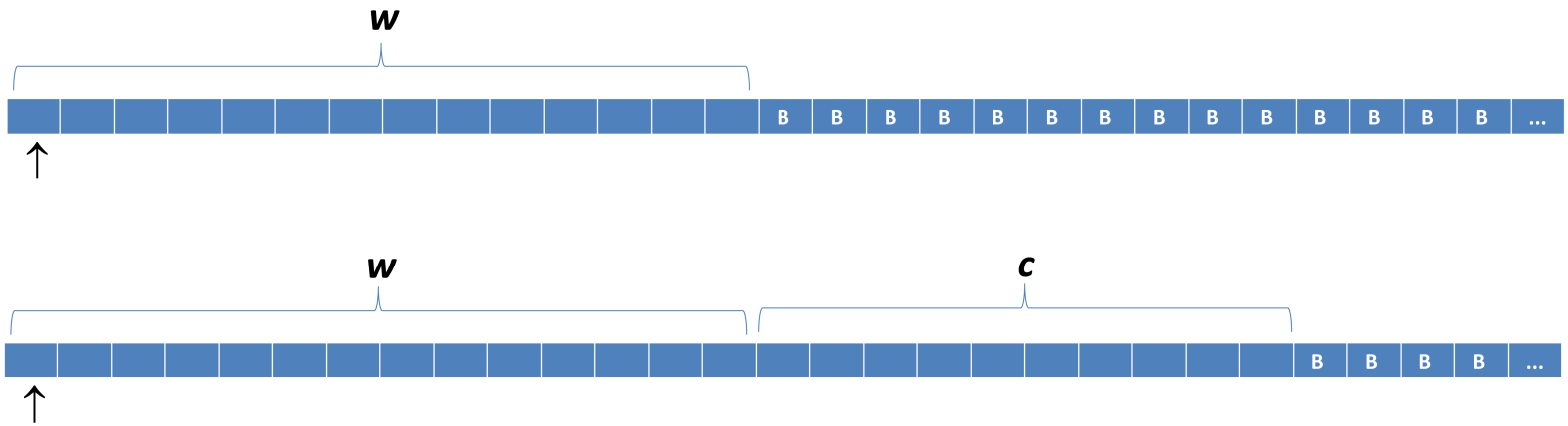
Definição

Um verificador para uma linguagem A é um algoritmo V tal que:

$$A = \{w \mid V \text{ aceita } \langle w, c \rangle \text{ para alguma cadeia } c\}$$

- O verificador apenas analisa uma possível solução para o problema e informa se ela de fato é uma solução (aceitando-a) ou não (rejeitando-a);
- O verificador não encontra uma solução para o problema, ele apenas verifica uma possível solução gerada à priori;
- Um verificador polinomial é aquele cujo tempo de execução é polinomial com o tamanho da cadeia w;
- Uma linguagem é polinomialmente verificável se ela tem um verificador de tempo polinomial.

Definição



Definição

Seja V um verificador polinomial. Então $|c| = O(|w|^k)$, pois:

- Essa é a quandidade máxima de símbolos que V pode analisar dentro da sua limitação de tempo.

Exemplos

Verificadores para algumas linguagens:

- Para $CAMHAM$, c é um caminho qualquer entre dois nós; o verificador apenas testa se esse caminho é um caminho Hamiltoniano;
- Para $COMPOSTOS$, c é um número inteiro qualquer; o verificador apenas testa se x é divisível por ele;

Definição alternativa

$\mathcal{NP}$ é a classe das linguagens que tem verificadores de tempo polinomial.

Equivalência das definições

Teorema:
L possui um verificador de tempo polinomial $\Leftrightarrow$ L é decidida por uma MT não-determinística de tempo polinomial.

MTs não-determinísticas de tempo polinomial $\times$ verificadores polinomiais

L possui um verificador V de tempo polinomial $\Rightarrow$ L é decidida por uma MT não-determinística N de tempo polinomial.

Prova

Sobre a entrada w de comprimento n:

- Suponha que V seja uma MT de tempo n^k;
- Construir uma MT não-determinística N tal que:
  1. N seleciona não-determinísticamente uma cadeia c de comprimento máximo n^k;
  2. N executa V sobre $\langle w, c \rangle$;
  3. Se V aceitar, N aceita; se V rejeitar, N rejeita.
- N gera todas as possíveis soluções e usa V para testar cada uma delas.

MTs não-determinísticas de tempo polinomial $\times$ verificadores polinomiais

L é decidida por uma MT não-determinística N de tempo polinomial $\Rightarrow$ L possui um verificador V de tempo polinomial.

Prova

Sobre a entrada $\langle w, c \rangle$:

- Considerar N como uma MT que decide L pelo método da força bruta (sempre existe uma);
- Construir V tal que:
  1. V simula N sobre a entrada w, usando os símbolos de c para orientar as suas escolhas não-determinísticas iniciais;
  2. Ela executa V sobre $\langle w, c \rangle$;
  3. Se N aceitar, V aceita; se N rejeitar, V rejeita.
- V se comporta como N, exceto por escolher a cadeia c ao invés de fazer escolhas não-determinísticas.

Definições

- $L \in \mathcal{NP} \Leftrightarrow L$ é decidida por uma MT não-determinística de tempo polinomial;
- $L \in \mathcal{NP} \Leftrightarrow L$ possui um verificador de tempo polinomial.

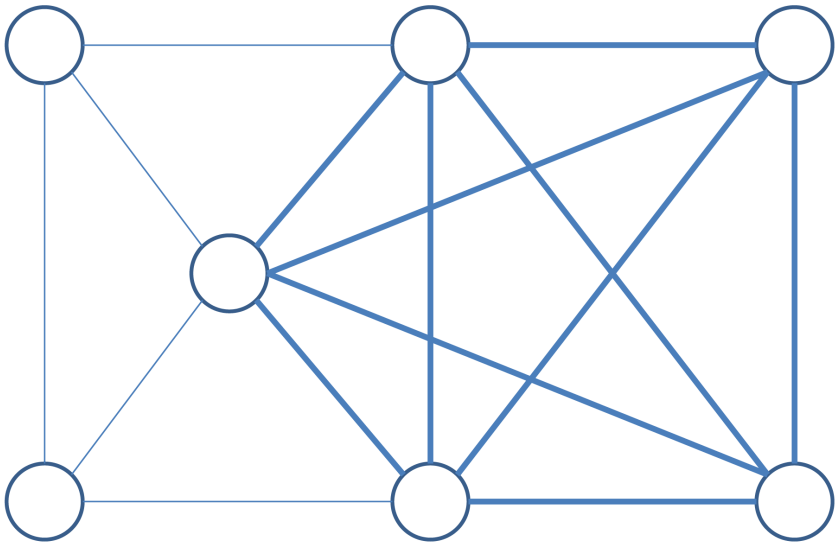
Problema $CLIQUE$

$CLIQUE = \{\langle G, k\rangle \mid G$ é um grafo não-direcionado com um $k - clique\}$

- Um $clique$ em um grafo não-direcionado é um subgrafo no qual todo par de nós está conectado por uma aresta;
- Um $k - clique$ é um $clique$ que contém k nós.

Problema $CLIQUE$

- Grafo com um $5-clique$:



- $CLIQUE \in \mathcal{NP}$ mas não se sabe se $CLIQUE \in \mathcal{P}$.

Problema $CLIQUE$

Teorema: $CLIQUE \in \mathcal{NP}$.

Prova usando uma MT não-determinística de tempo polinomial:

- Construir uma MT não-determinística N tal que, a partir da entrada $\langle G, k \rangle$:
  1. Não-determinísticamente seleciona um conjunto c formado por k nós;
  2. Verifica se G contém arestas que conectam todos os pares de nós de c;
  3. Em caso afirmativo, aceita; caso contrário, rejeita.

Problema $CLIQUE$

Teorema: $CLIQUE \in \mathcal{NP}$.

Prova usando um verificador de tempo polinomial:

- Construir um verificador V tal que, a partir da entrada $\langle\langle G, k\rangle, c\rangle$:
  1. Verifica se c é um conjunto de nós em G; rejeita caso contrário;
  2. Verifica se G contém arestas que conectam todos os pares de nós de c;
  3. Em caso afirmativo, aceita; caso contrário, rejeita.

Problema $SOMA_{SUBC}$

$$SOMA_{SUBC} = \{\langle S, t\rangle \mid S = \{x_1, x_2, ..., x_k\} \text{ e, para algum}$$

$$Y = \{y_1, ..., y_m\} \subseteq S, \sum_{i=1}^{m} y_i = t\}$$

- Sejam dadas uma coleção (conjunto em que são permitidas repetições de elementos) de números inteiros $S = \{x_1, x_2, ..., x_k\}$ e um número inteiro t;
- Determinar se existe uma coleção (idem) $Y = \{y_1, ..., y_m\} \subseteq S$ tal que $\sum_{i=1}^{m} y_i = t$.

Problema $SOMA_{SUBC}$

- $\langle \{4, 11, 16, 21, 2, 7\}, 25 \rangle \in SOMA_{SUBC}$ pois $4 + 21 = 25$;
  $\langle \{4, 11, 16, 21, 2, 7\}, 1 \rangle \notin SOMA_{SUBC}$ pois todos os elementos x_i de S são maiores do que 1;
- $SOMA_{SUBC} \in \mathcal{NP}$ mas não se sabe se $SOMA_{SUBC} \in \mathcal{P}$.

Problema $SOMA_{SUBC}$

Teorema: $SOMA_{SUBC} \in \mathcal{NP}$.

Prova usando uma MT não-determinística de tempo polinomial:

- Construir uma MT não-determinística N tal que, a partir da entrada $\langle S, t \rangle$:
  1. Não-determinísticamente seleciona um conjunto c dos números em S;
  2. Verifica se a soma dos números de c é igual à t;
  3. Em caso afirmativo, aceita; caso contrário, rejeita.

Problema $SOMA_{SUBC}$

Teorema: $SOMA_{SUBC} \in \mathcal{NP}$.

Prova usando um verificador de tempo polinomial:

- Construir um verificador V tal que, a partir da entrada $\langle\langle S, t\rangle, c\rangle$:
  1. Verifica se os elementos de c estão todos contidos em S;
  2. Verifica se a soma dos números de c é igual à t;
  3. Em caso afirmativo, aceita; caso contrário, rejeita.

Resumo

- $\mathcal{P}$ é o conjunto das linguagens que são decidíveis em tempo polinomial em MTs determinísticas;
- $\mathcal{P}$ é o conjunto das linguagens cuja pertinência pode ser decidida rapidamente;
- $\mathcal{NP}$ é o conjunto das linguagens que são decidíveis em tempo polinomial em MTs não-determinísticas (equivalentemente, em MTs determinísticas de tempo exponencial);
- $\mathcal{NP}$ é o conjunto das linguagens cuja pertinência pode ser decidida lentamente;
- $\mathcal{NP}$ é o conjunto das linguagens que são verificáveis em tempo polinomial;
- $\mathcal{NP}$ é o conjunto das linguagens cuja pertinência pode ser verificada rapidamente.

$\mathcal{P} \times \mathcal{NP}$

Como se relacionam os conjuntos $\mathcal{P}$ e $\mathcal{NP}$?

- Como toda linguagem que pode ser decidida por uma MT determinística pode também ser decidida por uma MT não-determinística, segue que $\mathcal{P} \subseteq \mathcal{NP}$;
- Resta saber se $\mathcal{P} = \mathcal{NP}$ ou se $\mathcal{P} \neq \mathcal{NP}$;
- Em outras palavras: será que todo algoritmo de tempo exponencial tem um equivalente de tempo polinomial? Ou será que existem problemas que só podem ser resolvidos através de algoritmos de tempo exponencial?
- A questão $\mathcal{P}$ versus $\mathcal{NP}$ é um dos maiores problemas ainda não resolvidos na Ciência da Computação e da Matemática.

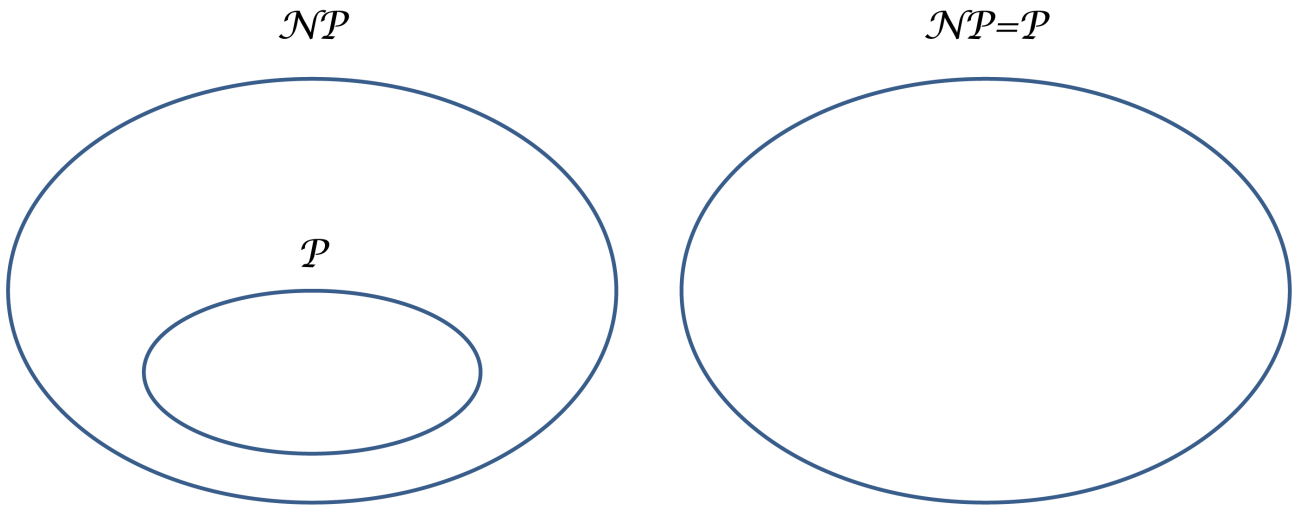
$\mathcal{P} \times \mathcal{NP}$

- Muita pesquisa e muitos esforços já foram dedicados para descobrir algoritmos de tempo polinomial para diversos problemas de tempo exponencial de interesse prático;
- Da mesma forma, para provar que existem certos problemas de tempo exponencial que não possuem algoritmos de tempo polinomial;
- No entanto, nenhum resultado concreto foi obtido até hoje para nenhuma dessas iniciativas;
- A impressão geral é que existem problemas que só podem ser resolvidos por algoritmos exponenciais, ou seja, que $\mathcal{P} \neq \mathcal{NP}$.

$\mathcal{P} \times \mathcal{NP}$

- Se algum dia for demonstrado que $\mathcal{P} = \mathcal{NP}$, então todos os problemas de tempo exponencial poderão ser decididos em tempo polinomial;
- Se algum dia for demonstrado que $\mathcal{P} \neq \mathcal{NP}$, então saberemos que existem problemas intrinsecamente difíceis (aqueles pertencentes à $\mathcal{NP} - \mathcal{P}$), para os quais todas as soluções algoritmicas serão necessariamente exponenciais;
- O Clay Mathematics Institute oferece US$1.000.000,00 para quem apresentar uma prova formal de que $\mathcal{P} \neq \mathcal{NP}$ ou $\mathcal{P} = \mathcal{NP}$;
- `http://www.claymath.org/millennium-problems/p-vs-np-problem`

$\mathcal{P} \times \mathcal{NP}$



Apenas uma das alternativas acima é a correta, mas ainda não se sabe qual delas.

Perguntas

Dada uma linguagem decidível L qualquer, como provar:

- $L \in \mathcal{P}$?
  - Apresentando uma MT determinística de tempo polinomial que decide L, ou
  - Efetuando uma redução para outro problema pertencente à $\mathcal{P}$ (conforme explicado mais adiante).
- $L \in \mathcal{NP}$?
  - Apresentando uma MT não-determinística de tempo polinomial que decide L, ou
  - Apresentando um verificador de tempo polinomial para L.
- $L \notin \mathcal{P}$?
  - Não existe método conhecido. Pode-se apenas exibir fortes evidências de que isso seja verdade (através da NP-completude, explicada mais adiante).

Conceito

- A redutibilidade estudada anteriormente não leva em consideração o tempo necessário para mapear instâncias de um problema P_1 em instâncias de um problema P_2;
- Uma nova definição será feita, de forma a garantir que a redução seja feita de forma "eficiente";
- Dessa maneira, uma solução eficiente para um problema P_2 poderá ser usada para resolver, de forma igualmente eficiente, um problema P_1.

Função computável em tempo polinomial

Uma função $f : \Sigma^* \to \Sigma^*$ é dita uma "função computável em tempo polinomial" se existir uma MT determinística de tempo polinomial que, ao processar a entrada w, pára com exatamente $f(w)$ gravada na sua fita.

Redução em tempo polinomial

Uma linguagem A é dita "redutível em tempo polinomial" à linguagem B, denotado $A \leq_P B$, se existir uma função computável em tempo polinomial $f : \Sigma^* \to \Sigma^*$ tal que:

$$w \in A \Leftrightarrow f(w) \in B$$

Diz-se que a função f é uma "redução de tempo polinomial" de A para B.

Redução em tempo polinomial

Se uma linguagem A é redutível em tempo polinomial para uma outra linguagem B, sabidamente decidível em tempo polinomial por uma MT determinística, segue que existe uma MT determinística de tempo polinomial que decide A.

Reduções e decidibilidade/complexidade

Decidibilidade:

- Uma solução para um problema B, combinada com uma redução (qualquer) de A para B, produz uma solução para A.

Complexidade:

- Uma solução eficiente para um problema B, combinada com uma redução eficiente de A para B, produz uma solução eficiente para A.

Teorema 1

Teorema:
Se $A \leq_P B$ e $B \in \mathcal{P}$, então $A \in \mathcal{P}$.
Prova:

- Seja M a MT que decide B;
- Seja f a redução de tempo polinomial de A para B;
- Construir N tal que, para a entrada w:
  1. N computa $f(w)$;
  2. N simula M com a entrada $f(w)$;
  3. Se M aceita, aceitar; se M rejeita, rejeitar.
- Como f executa em tempo polinomial e M executa em tempo polinomial, segue que a composição, ou seja N, também executa em tempo polinomial e portanto A é decidida em tempo polinomial, ou seja, $A \in \mathcal{P}$.

Teorema 1

De Hopcroft07:

- Suponha $w \in A$, $|w| = m$;
- Como $f(w)$ é $O(m^j)$, para algum j, então $|f(w)| = c \cdot m^j$ para algum c;
- Suponha que M, que decide B, é $O(n^k)$;
- Então, o tempo que M leva para decidir $f(w)$ é $O((c \cdot m^j)^k)$;
- O tempo total consumido, incluindo o mapeamento de w em $f(w)$, é $O(m^j) + O((c \cdot m^j)^k) = O(m^j + (c \cdot m^j)^k)$;
- Simplificando, temos $O(m^j + c^k \cdot m^{jk})$;
- Como c, j e k são constantes, segue que o tempo total para N é polinomial em função de m.

Teorema 1

"Reduções de tempo polinomial são importantes pois o limite imposto no número de transições executadas pela máquina que efetua a redução limita também o comprimento da cadeia que é gerada para a máquina decisora. Essa propriedade garante que a combinação de uma redução de tempo polinomial com um algoritmo de tempo polinomial produz um outro algoritmo de tempo polinomial."

Teorema 2

Teorema: Se $A \leq_P B$ e $B \leq_P C$, então $A \leq_P C$.
Prova:

- A composição de reduções polinomiais é também uma redução polinomial.

A relação "redução em tempo polinomial" é transitiva.
Detalhes em Martin91.

Fórmula booleana

- Variável boolena: aquela que pode assumir os valores "verdadeiro" (1) ou "falso" (0);
- Operações boolenas: "E" ($\wedge$), "OU" ($\vee$) e "NÃO" (barra superior):

$$\begin{array}{l|l|l} 0 \wedge 0 = 0 & 0 \vee 0 = 0 & \overline{0} = 1 \\ 0 \wedge 1 = 0 & 0 \vee 1 = 1 & \overline{1} = 0 \\ 1 \wedge 0 = 0 & 1 \vee 0 = 1 & \\ 1 \wedge 1 = 1 & 1 \vee 1 = 1 & \end{array}$$

- Fórmula booleana: composta por variáveis e operações booleanas. Exemplo: $\phi = (\overline{x} \wedge y) \vee (x \wedge \overline{z})$

Problema SAT

- Uma fórmula booleana é dita 'satisfazível" se houver alguma atribuição de valores 0 e 1 para as suas variáveis de tal forma que o valor resultante seja 1;
  Exemplo: a fórmula $\phi = (\overline{x} \wedge y) \vee (x \wedge \overline{z})$ é satisfazível pois $x = 0, y = 1, z = 0$ faz com que $\phi = 1$;
- $SAT = \{\langle \phi \rangle \mid \phi \text{ é uma fórmula satisfazível}\}$;
- SAT é decidível: basta testar se alguma das 2^n possíveis combinações de valores para as n variáveis da fórmula satisfaz à mesma;
- $SAT \in \mathcal{NP}$;
- $SAT \in \mathcal{P}$?

FNC-fórmula

- Literal: variável booleana ou variável booleana negada;
  Exemplos: x_1, $\overline{x_2}$
- Cláusula: fórmula composta por literais e operadores "OU" ($\vee$);
  Exemplo: $(x_1 \vee \overline{x_2} \vee \overline{x_3} \vee x_4)$
- FNC-fórmula: (de *Forma Normal Conjuntiva*) fórmula composta por cláusulas conectadas por operadores "E" ($\wedge$);
  Exemplo: $(x_1 \vee \overline{x_2} \vee \overline{x_3} \vee x_4) \wedge (x_3 \vee \overline{x_5} \vee x_6) \wedge (x_3 \vee \overline{x_6})$
- Prova-se que toda fórmula boolena pode ser convertida para uma FNC-fórmula equivalente.

Problema $3SAT$

- Caso especial do problema SAT;
- $3FNC$-fórmula: FNC-fórmula em que todas as cláusulas contém exatamente três literais;
  Exemplo: $(x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (x_3 \vee \overline{x_5} \vee x_6) \wedge (x_3 \vee x_5 \vee x_6)$
- $3SAT = \{\langle\phi\rangle \mid \phi$ é uma $3FNC$-fórmula satisfazível$\}$;
  Exemplo: $\phi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2) \in 3SAT$ pois ela tem valor 1 com $x_1 = 0$ e $x_2 = 1$;
- Para a fórmula ser satisfazível, cada cláusula deve conter um literal com o valor 1;
- Prova-se que toda fórmula booleana possui uma $3FNC$-fórmula equivalente;
- O tamanho desta fórmula, no entanto, cresce exponencialmente com o tamanho da entrada.

Redução de $3SAT$ para $CLIQUE$

Teorema: $3SAT$ é redutível em tempo polinomial para $CLIQUE$.
Prova:

- Seja $\phi = (a_1 \vee b_1 \vee c_1) \wedge (a_2 \vee b_2 \vee c_2) \wedge ...(a_k \vee b_k \vee c_k)$
- A redução produz uma cadeia $\langle G, k \rangle$, onde G é um grafo não-direcionado com um k-clique sse ϕ é satisfazível;
- A construção de G é apresentada a seguir.

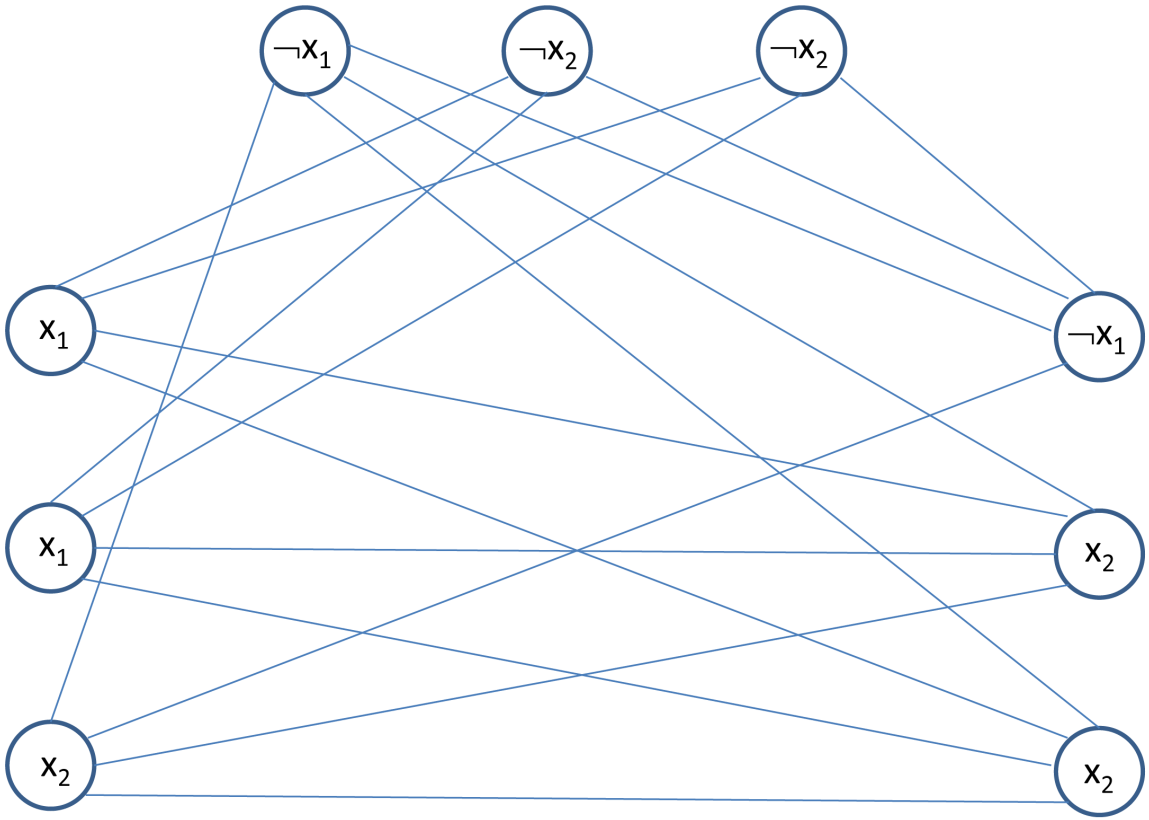
Redução de $3SAT$ para $CLIQUE$

Construção de G a partir de ϕ:

- G possui $k \cdot 3$ nós;
- Cada nó representa um literal de ϕ;
- Os nós são agrupados em triplas, cada tripla corresponde a uma cláusula;
- Os arcos conectam todos os pares de nós de G, exceto:
  - Nós que fazem parte da mesma tripla (exemplo: a_1 e b_1);
  - Nós que representam um literal e a sua negação (exemplo: a_1 e $\overline{a_1}$).

Redução de $3SAT$ para $CLIQUE$

Exemplo para $\phi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$:



Estratégias para provar a redução

Suponha que se deseje provar:

$$w \in A \text{ se e somente se } f(w) \in B$$

$$(w \in A \Leftrightarrow f(w) \in B)$$

Duas estratégias são possíveis para obter a prova:

1. $w \in A \Rightarrow f(w) \in B$
   $w \notin A \Rightarrow f(w) \notin B$
2. $w \in A \Rightarrow f(w) \in B$
   $f(w) \in B \Rightarrow w \in A$

Elas são equivalentes. Usaremos a segunda.

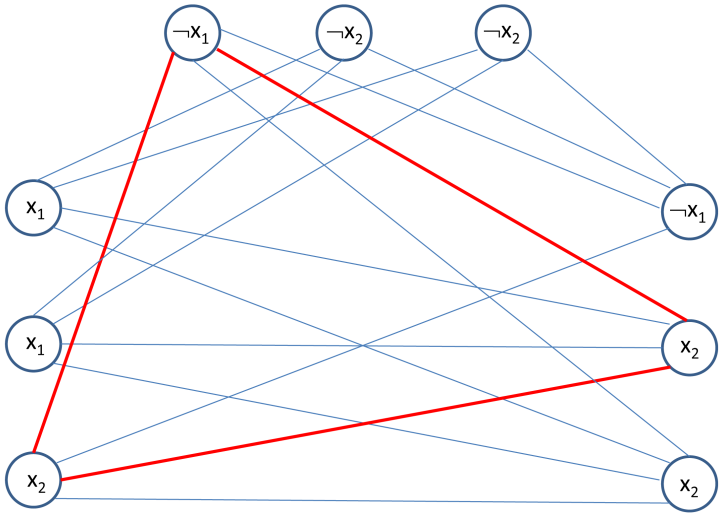
Redução de $3SAT$ para $CLIQUE$

ϕ é satisfazível $\Rightarrow$ G tem um k-clique:

- Seja $\phi = (a_1 \vee b_1 \vee c_1) \wedge (a_2 \vee b_2 \vee c_2) \wedge ...(a_k \vee b_k \vee c_k)$;
- Se ϕ é satisfazível, então pelo menos um literal é verdadeiro em cada cláusula;
- Selecionar o nó correspondente em cada tripla (se houver mais de um, fazer uma seleção arbitrária);
- O número de nós selecionado é k, pois foi escolhido um nó em cada tripla;
- Os nós selecionados estão todos interligados, pois:
  - Nenhum par é da mesma tripla (foi escolhido apenas um literal em cada cláusula);
  - Nenhum par é formado por nós contraditórios (um literal e o seu complemento não podem ser ambos verdadeiros).
- Logo, G tem um k-clique.

Redução de $3SAT$ para $CLIQUE$

$\langle \phi \rangle \in SAT \Rightarrow \langle G, 3 \rangle \in CLIQUE$



- $\phi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$ é satisfazível para $x_1 = 0$ e $x_2 = 1$;
- G tem um 3-clique formado pelos nós $x_2, \overline{x_1}, x_2$.

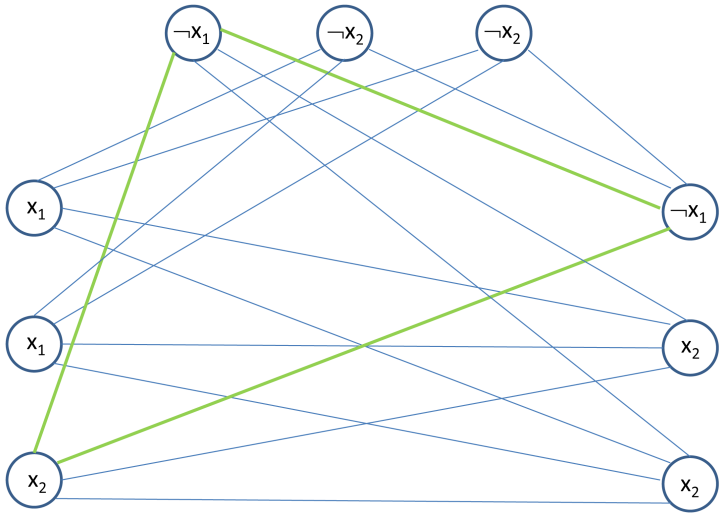
Redução de $3SAT$ para $CLIQUE$

G tem um k-clique $\Rightarrow$ ϕ é satisfazível:

- Seja G um grafo com um k-clique;
- Nenhum par de nós desse k-clique pertence à mesma tripla, pois nós da mesma tripla não são conectados;
- Logo, cada uma das k triplas possui exatamente um nó no k-clique;
- Selecionar os nós do k-clique e atribuir valores às variáveis de tal forma que os correspondentes literais tenham o valor verdadeiro;
- Como nós contraditórios não são conectados, essa atribuição será sempre possível;
- A atribuição satisfaz ϕ, pois cada cláusula possui um literal com o valor verdadeiro.
- Logo, ϕ é satisfazível.

Redução de $3SAT$ para $CLIQUE$

$\langle G, 3\rangle \in CLIQUE \Rightarrow \langle \phi \rangle \in SAT$



- G tem um 3-clique formado pelos nós $x_2, \overline{x_1}, \overline{x_1}$
- Selecionar $x_2, \overline{x_1}, \overline{x_1}$ e fazer $x_2 = 1$ e $x_1 = 0$.
  $\phi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$ é satisfazível.

Redução de $3SAT$ para $CLIQUE$

Seja:

$$\phi = (x_1 \vee x_1 \vee x_1) \wedge (\overline{x_1} \vee \overline{x_1} \vee \overline{x_1}) \wedge (x_2 \vee x_2 \vee x_2) \wedge (\overline{x_2} \vee \overline{x_2} \vee \overline{x_2})$$

- ϕ não é satisfazível ($\phi \notin SAT$), pois pelo menos uma cláusula é sempre falsa;
- Portanto, G não tem um 4-clique e $\langle G, 4 \rangle \notin CLIQUE$;
- Não obstante, é fácil verificar que $\langle G, 2 \rangle \in CLIQUE$ (usando, por exemplo, os nós x_1 e x_2).

Redução de $3SAT$ para $CLIQUE$

Análise do tempo de execução do algoritmo proposto:

- Seja $\phi = (a_1 \vee b_1 \vee c_1) \wedge (a_2 \vee b_2 \vee c_2) \wedge ...(a_k \vee b_k \vee c_k)$;
- ϕ possui $3 \cdot k$ literais (distintos ou não);
- G possui $3 \cdot k$ nós distintos;
- Para determinar os arcos de G, é necessário inspecionar os $3 \cdot k$ nós individualmente e, para cada um deles analisar possíveis conexões com outros $3 \cdot k - 3$ nós (todos exceto os da mesma tripla);
- Portanto, são necessárias $(3 \cdot k) \cdot (3 \cdot k - 3) = 9 \cdot k^2 - 9 \cdot k$ análises distintas;
- Logo, o algoritmo de redução possui tempo polinomial com o tamanho da entrada.

Redução de $3SAT$ para $CLIQUE$

- Temos uma redução de tempo polinomial de $3SAT$ para $CLIQUE$;
- Se $CLIQUE$ for solucionável em tempo polinomial, então $3SAT$ também o será;
- Problemas de naturezas completamente diferentes, como $3SAT$ e $CLIQUE$, possuem complexidades relacionadas;
- Esse resultado pode ser generalizado para toda uma classe de problemas.

Conceitos

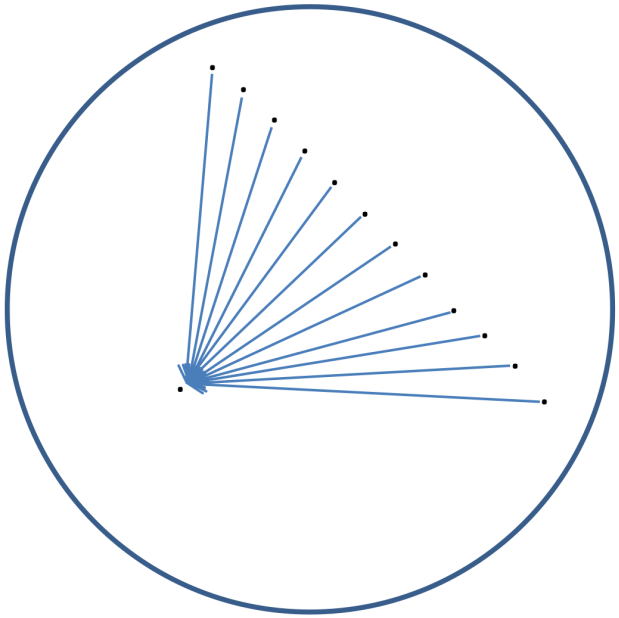
- A complexidade de certos problemas em $\mathcal{NP}$ está relacionados com a complexidade de todos os problemas de $\mathcal{NP}$;
- Esses problemas formam um subconjunto de $\mathcal{NP}$;
- A sua descoberta, na década 70, foi um marco na história da Ciência da Computação.

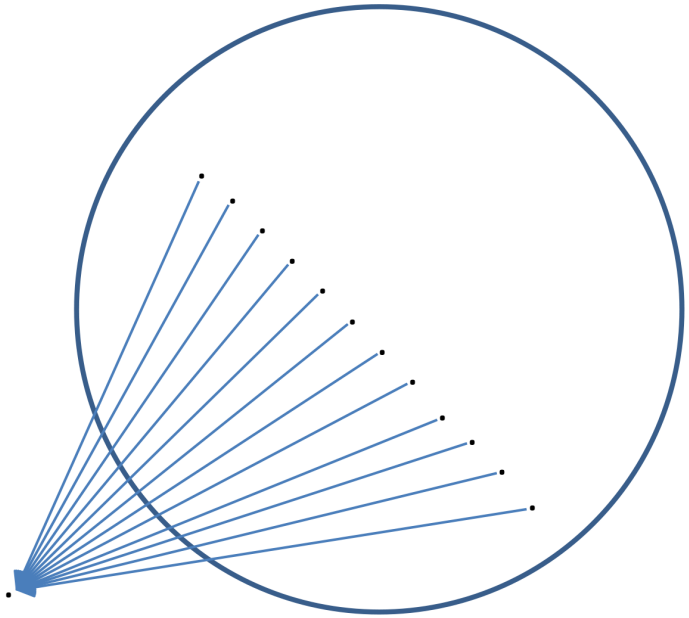
Problema $\mathcal{NP}$-hard

Um problema (linguagem) B é dito $\mathcal{NP}$-hard se a seguinte condição for verificada:

- B não necessita pertencer à $\mathcal{NP}$;
- $\forall A \in \mathcal{NP}$, existe uma redução de tempo polinomial de A para B.

Problema $\mathcal{NP}$-hard





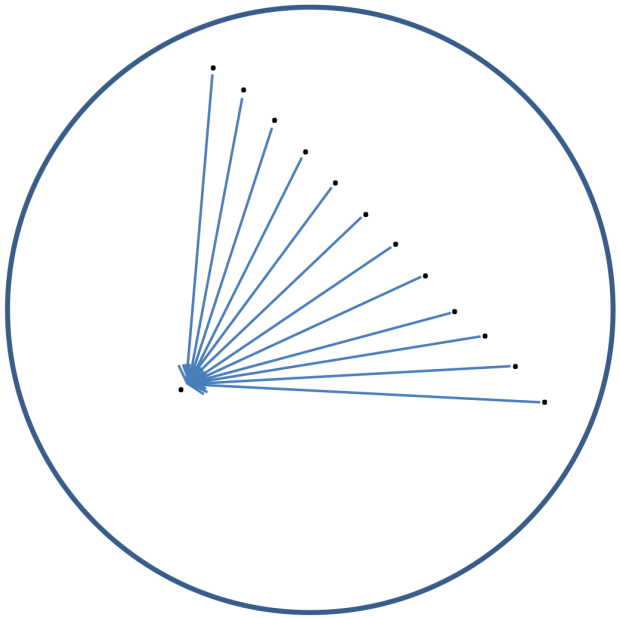
B pode ou não pertencer à $\mathcal{NP}$

Problema $\mathcal{NP}$-completo

Um problema (linguagem) B é dito $\mathcal{NP}$-completo se as seguintes condições forem verificadas:

- $B \in \mathcal{NP}$;
- $\forall A \in \mathcal{NP}$, existe uma redução de tempo polinomial de A para B.

Problema $\mathcal{NP}$-completo



B deve pertencer à $\mathcal{NP}$

Problema $\mathcal{NP}$-completo

Ou seja:

- Um problema $\mathcal{NP}$-completo é um problema $\mathcal{NP}$-hard que pertence à $\mathcal{NP}$;
- Todo problema $\mathcal{NP}$-completo é também um problema $\mathcal{NP}$-hard;
- Nem todo problema $\mathcal{NP}$-hard é $\mathcal{NP}$-completo;
- Um problema B é $\mathcal{NP}$-hard sse existir algum problema $\mathcal{NP}$-completo A que seja redutível para B.

Problema $\mathcal{NP}$-completo

Problemas $\mathcal{NP}$-completos podem ser considerados como "problemas universais" na classe $\mathcal{NP}$:

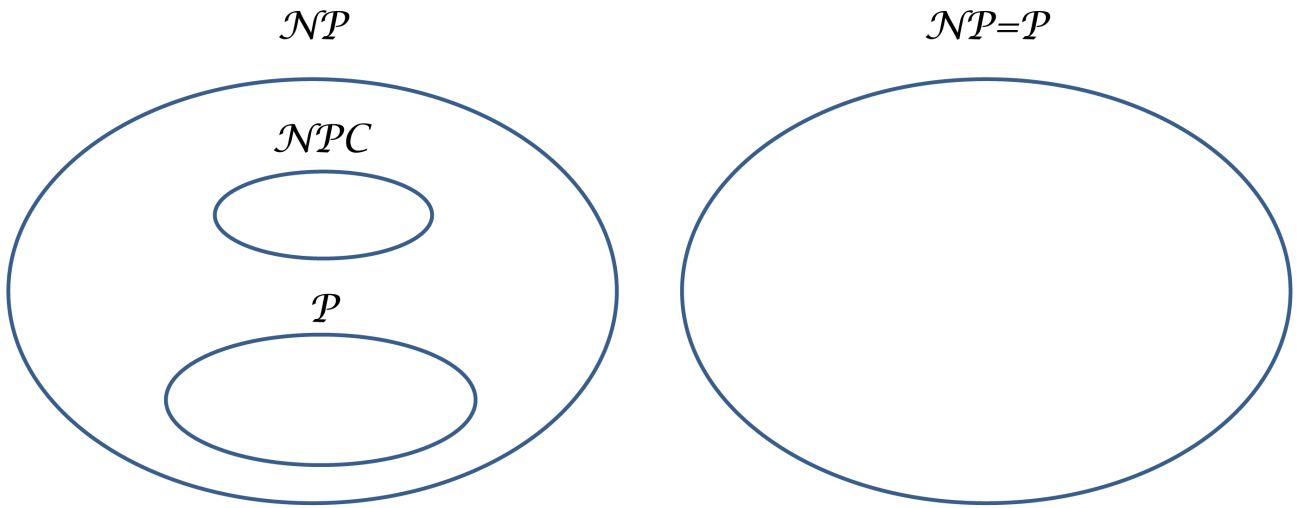
- Eles são pelo menos tão difíceis quanto qualquer outro problema em $\mathcal{NP}$;
- A eventual descoberta de uma máquina determinística de tempo polinomial que resolva um tal problema é suficiente para permitir a construção de máquinas com as mesmas características que resolvam todos os problemas em $\mathcal{NP}$;
- Tal descoberta provaria que $\mathcal{P} = \mathcal{NP}$.

As três considerações acima são válidas também para problemas $\mathcal{NP}$-hard.

A classe $\mathcal{NPC}$

- A classe $\mathcal{NPC}$ é formada por todos os problemas $\mathcal{NP}$-completos;
- $\mathcal{NPC} \subseteq \mathcal{NP}$;
- A complexidade de todos os problemas em $\mathcal{NP}$ é relacionada individualmente com a complexidade de cada problema em $\mathcal{NPC}$.
- Problemas $\mathcal{NP}$-completos aparecem em todas as áreas;
- A maioria dos problemas $\mathcal{NP}$ conhecidos está em $\mathcal{P}$ ou $\mathcal{NPC}$;
- Se um problema está em $\mathcal{NPC}$, essa é uma forte evidência de que provavelmente ele não está em $\mathcal{P}$;
- Problemas $\mathcal{NP}$-completos são os principais candidatos para estar em $\mathcal{NP} - \mathcal{P}$.

A classe $\mathcal{NPC}$



Apenas uma das alternativas acima é a correta, mas ainda não se sabe qual delas.

Teorema 3

Teorema: Se $B \in \mathcal{NPC}$ e $B \in \mathcal{P}$, então $\mathcal{P} = \mathcal{NP}$.
Prova:

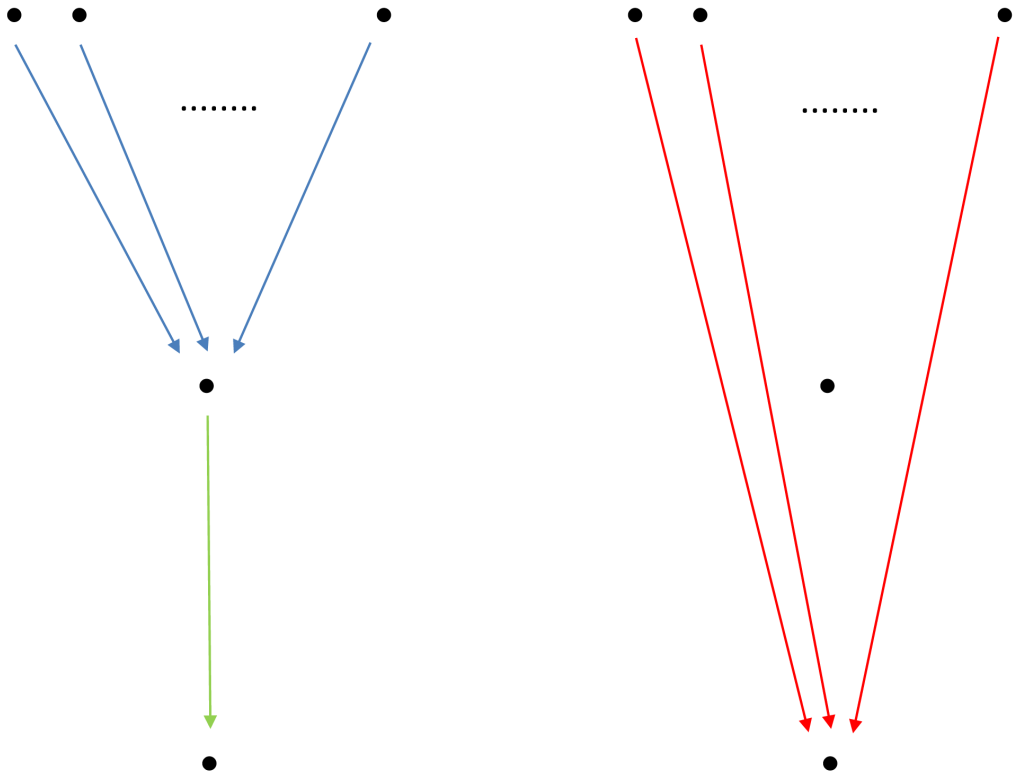
- Decorre diretamente da definição de redutibilidade em tempo polinomial.

Teorema 4

Teorema: Se $B \in \mathcal{NPC}$ e $B \leq_P C$, com $C \in \mathcal{NP}$, então $C \in \mathcal{NPC}$.
Prova:

- Como $B \in \mathcal{NPC}$, então todos os problemas de $\mathcal{NP}$ reduzem em tempo polinomial para B;
- Como B reduz em tempo polinomial para C, e a composição de reduções polinomiais é também polinomial, então todos os problemas de $\mathcal{NP}$ são redutíveis em tempo polinomial para C;
- Como, por suposição, $C \in \mathcal{NP}$, então $C \in \mathcal{NPC}$.

Teorema 4



Teorema 5

Teorema: Se $B \in \mathcal{NPH}$ e $B \leq_P C$, então $C \in \mathcal{NPH}$.
Prova:

- Como $B \in \mathcal{NPH}$, então todos os problemas de $\mathcal{NP}$ reduzem em tempo polinomial para B;
- Como B reduz em tempo polinomial para C, e a composição de reduções polinomiais é também polinomial, então todos os problemas de $\mathcal{NP}$ são redutíveis em tempo polinomial para C;
- Logo, $C \in \mathcal{NPC}$.

$SAT \in \mathcal{NPC}$

Teorema: $SAT \in \mathcal{NPC}$.
Prova:

- Conhecido como Teorema de Cook (ou Cook-Levin), foi provado em 1971;
- A prova é longa e repleta de detalhes (Hopcroft-2007);
- Ela é baseada em nas seguintes ideias:
  - Demonstração de que $SAT \in \mathcal{NP}$ (simples): para isso, é suficiente apresentar uma MT não-determinística que adivinha uma atribuição de valores booleanos para as variáveis e depois testa se essa atribuição satisfaz à fórmula.
  - Demonstração de que todos os problemas $A \in \mathcal{NP}$ possuem uma redução de tempo polinomial para SAT (complexa): é feita através de uma redução que simula a MT para A com a entrada w, produzindo uma fórmula booleana ϕ tal que $w \in A \Leftrightarrow \phi$ é satisfazível.

$SAT \in \mathcal{NPC}$

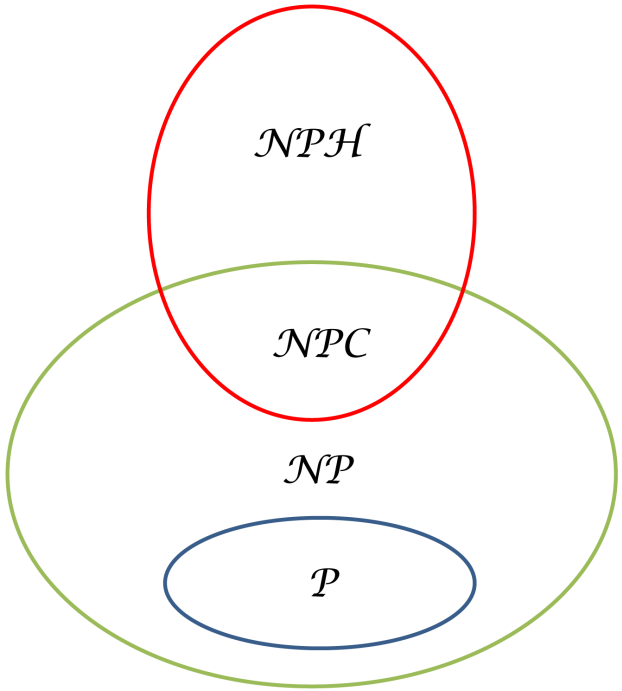
Corolário: $3SAT \in \mathcal{NPC}$.
Prova:

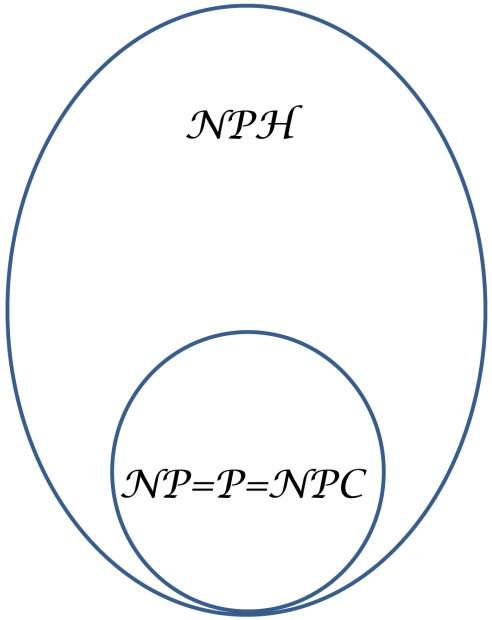
- $3SAT \in \mathcal{NP}$;
- Pode-se apresentar uma redução de tempo polinomial de SAT para $3SAT$, ou fazer uma demonstração direta, modificando a demonstração de que $SAT \in \mathcal{NPC}$;
- A redução de SAT para $3SAT$ mapeia fórmulas que são satisfazíveis simultaneamente, porém não necessariamente equivalentes.

Corolário: $CLIQUE \in \mathcal{NPC}$.
Prova:

- $CLIQUE \in \mathcal{NP}$;
- A redução de tempo polinomial de $3SAT$ para $CLIQUE$ apresentada anteriormente é a prova.

A classe $\mathcal{NPH}$





Apenas uma das alternativas acima é a correta, mas ainda não se sabe qual delas.

A classe $\mathcal{NPH}$

Exemplo de problema que pertence à $\mathcal{NPH}$ mas não pertence à $\mathcal{NPC}$:

$$PARA_{MT} = \{\langle M, w\rangle \mid M \text{ é uma MT que pára com a entrada } w\}$$

Prova:

- Redução de tempo polinominal de SAT para $PARA_{MT}$:
  - A partir de uma fórmula booleana ϕ, construir uma MT M que gera e testa todas as possíveis atribuições de valores para as suas variáveis;
  - Se alguma combinação de valores satisfizer ϕ, M pára; caso contrário, M entra em loop infinito.
- $PARA_{MT} \notin \mathcal{NP}$, pois $PARA_{MT}$ não é decidível;

A classe $\mathcal{NPH}$

Se existisse solução de tempo polinomial para um problema indecidível ($PARA_{MT}$), então existiria um algoritmo de tempo polinomial para um problema decidível (SAT) para o qual, pelo menos até hoje, se conhece apenas soluções de tempo exponencial.

Estratégias

Para provar que $\mathcal{P} = \mathcal{NP}$:

- Provar que algum (qualquer um) problema $\mathcal{NP}$-completo pertence à $\mathcal{P}$;
- Como conseqüência, todos os problemas de $\mathcal{NP}$ poderão ser resolvidos de forma eficiente.

Para provar que $\mathcal{P} \neq \mathcal{NP}$:

- Provar que existe pelo menos um problema A pertencente à $\mathcal{NP}$ que não pertence à $\mathcal{P}$;
- Se isso for provado, todos os problemas $\mathcal{NP}$-completos estarão automaticamente em $\mathcal{NP}$-$\mathcal{P}$ (pois, se assim não fosse, haveria um algoritmo eficiente para resolver A, e isso contradiz o fato de que $A \notin \mathcal{P}$).

Estratégias

Ao encontrar um novo problema:

- Verificar se algum problema $\mathcal{NP}$-completo é redutível à ele em tempo polinomial:
  - Em caso afirmativo, é improvável que exista algoritmo de tempo polinomial para ele (pois se houvesse haveria também para inúmeros outros problemas que já vem sendo pesquisados há anos) e provavelmente não vale à pena dedicar esforços nesse sentido.
- Verificar se esse novo problema é redutível em tempo polinomial para algum problema pertencente à $\mathcal{P}$:
  - Em caso afirmativo, o novo problema também pertence à $\mathcal{P}$.
- Pesquisar, diretamente, se esse novo problema pertence à $\mathcal{P}$.

Outros problemas $\mathcal{NP}$-completos

A demonstração de que um problema está em $\mathcal{NPC}$ pode ser feita:

- De forma direta (pouco utilizada), como no caso do SAT;
- Por redução de tempo polinomial a partir de algum outro problema reconhecidamente em $\mathcal{NPC}$ (largamente utilizada), como por exemplo SAT, $3SAT$ (principalmente) ou $CLIQUE$;
- Desde o início da década de 1970, milhares de problemas $\mathcal{NP}$-completos já foram identificados, todos eles de grande interesse prático;
- `https://en.wikipedia.org/wiki/List_of_NP-complete_problems`

Exemplos de problemas $\mathcal{NP}$-completos

1. *Problema:*
   Conjuntos independentes (CI).
2. *Descrição:*
   Determinar se um grafo G possui um subconjunto de nós tal que nenhum par de nós desse subconjunto esteja conectado por algum arco.
3. *Entrada:*
   Um grafo G e um número inteiro k entre 1 e o número de nós de G.
4. *Saída:*
   SIM se e somente se G tem um conjunto independente formado por k nós.
5. *Redução:*
   $3SAT$.

Exemplos de problemas $\mathcal{NP}$-completos

1. *Problema:*
   Cobertura de nós (CN).
2. *Descrição:*
   Determinar se um grafo G possui um subconjunto de nós tal que todos os arcos de G sejam tocados, em pelo menos uma extremidade, por algum nó do subconjunto.
3. *Entrada:*
   Um grafo G e um número inteiro k entre 0 e o número de nós de G menos um.
4. *Saída:*
   SIM se e somente se G tem um conjunto independente formado por k nós.
5. *Redução:*
   Conjuntos independentes.

Exemplos de problemas $\mathcal{NP}$-completos

"Circuito Hamiltoniano" em um grafo G:

1. Caminho que visita todos os nós de G uma única vez;
2. Deve formar um ciclo, ou seja, começar e terminar no mesmo nó;
3. A quantidade de nós do caminho é igual à quantidade de nós do grafo;
4. A quantidade de arcos do caminho é igual à quantidade de nós do grafo.

Não confundir "Caminho Hamiltoniano" (cada nó comparece uma única vez) com "Circuito Hamiltoniano" (idem, exceto que o primeiro nó deve ser igual ao último).

Exemplos de problemas $\mathcal{NP}$-completos

1. *Problema:*
   Circuito Hamiltoniano direcionado (CIHD).
2. *Descrição:*
   Determinar se um grafo direcionado G possui um Circuito Hamiltoniano.
3. *Entrada:*
   Um grafo direcionado G.
4. *Saída:*
   SIM se e somente se G possuir um Circuito Hamiltoniano.
5. *Redução:*
   $3SAT$.

Exemplos de problemas $\mathcal{NP}$-completos

1. *Problema:*
   Circuito Hamiltoniano não-direcionado (CIHN).
2. *Descrição:*
   Determinar se um grafo não-direcionado G possui um Circuito Hamiltoniano.
3. *Entrada:*
   Um grafo não-direcionado G.
4. *Saída:*
   SIM se e somente se G possuir um Circuito Hamiltoniano.
5. *Redução:*
   Circuito Hamiltoniano direcionado.

Exemplos de problemas $\mathcal{NP}$-completos

1. *Problema:*
   Caminho Hamiltoniano direcionado (CAHD).
2. *Descrição:*
   Determinar se um grafo direcionado G possui um Caminho Hamiltoniano.
3. *Entrada:*
   Um grafo direcionado G.
4. *Saída:*
   SIM se e somente se G possuir um Caminho Hamiltoniano.
5. *Redução:*
   $3SAT$.

Exemplos de problemas $\mathcal{NP}$-completos

1. *Problema:*
   Caminho Hamiltoniano não-direcionado (CAHN).
2. *Descrição:*
   Determinar se um grafo não-direcionado G possui um Caminho Hamiltoniano.
3. *Entrada:*
   Um grafo não-direcionado G.
4. *Saída:*
   SIM se e somente se G possuir um Caminho Hamiltoniano.
5. *Redução:*
   Caminho Hamiltoniano direcionado.

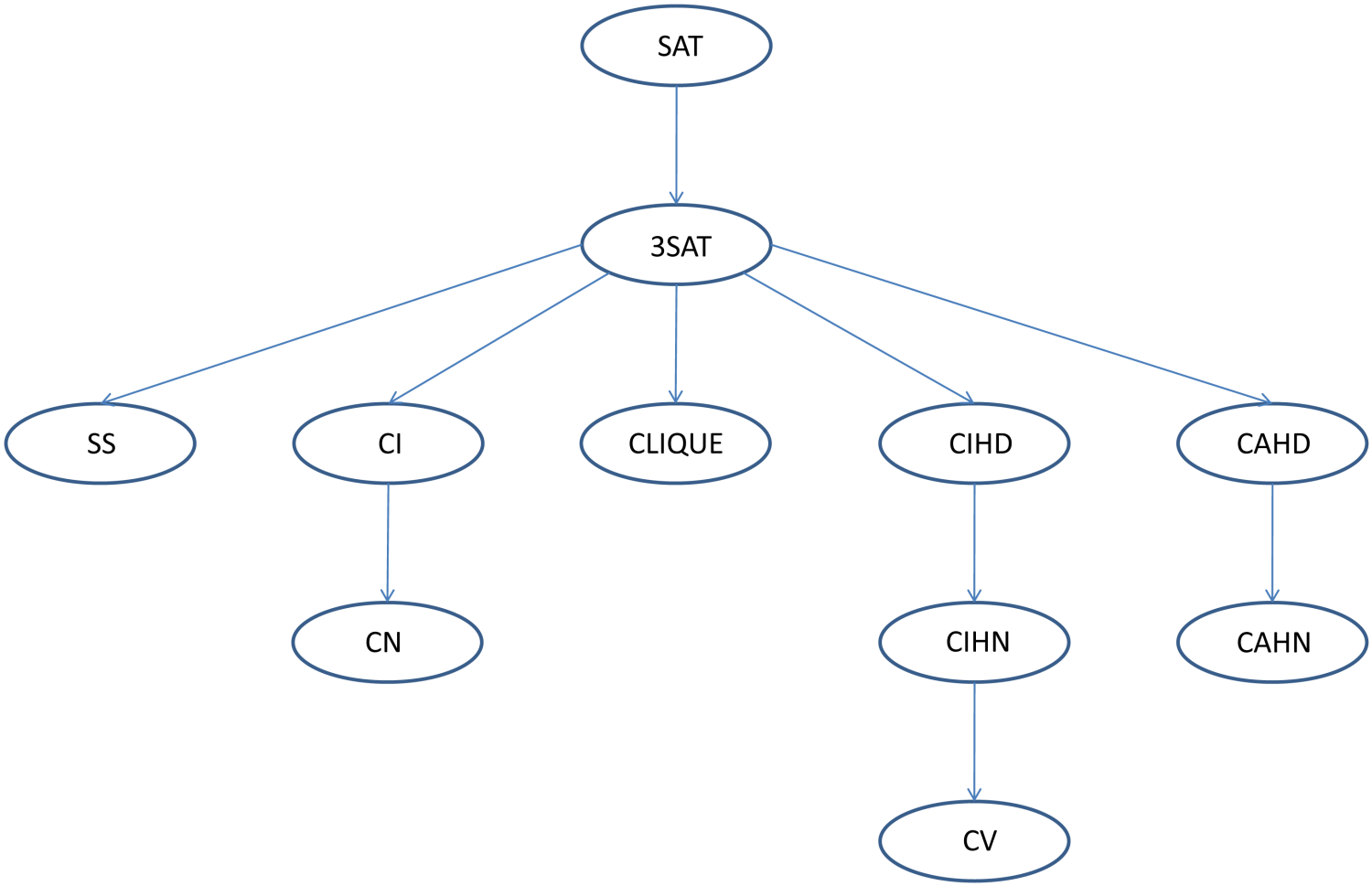
Exemplos de problemas $\mathcal{NP}$-completos

1. *Problema:*
   Caixeiro viajante (CV).
2. *Descrição:*
   Determinar se um grafo não-direcionado G, com valores númericos inteiros atribuídos aos seus arcos, possui um Circuito Hamiltoniano e, além disso, a soma dos pesos dos arcos que compõem esse caminho é menor ou igual a k.
3. *Entrada:*
   Um grafo não-direcionado G com valores inteiros atribuídos aos seus nós e um valor de k;
4. *Saída:*
   SIM se e somente se G possuir um Circuito Hamiltoniano com a caracerística descrita.
5. *Redução:*
   Circuito Hamiltoniano não-direcionado.

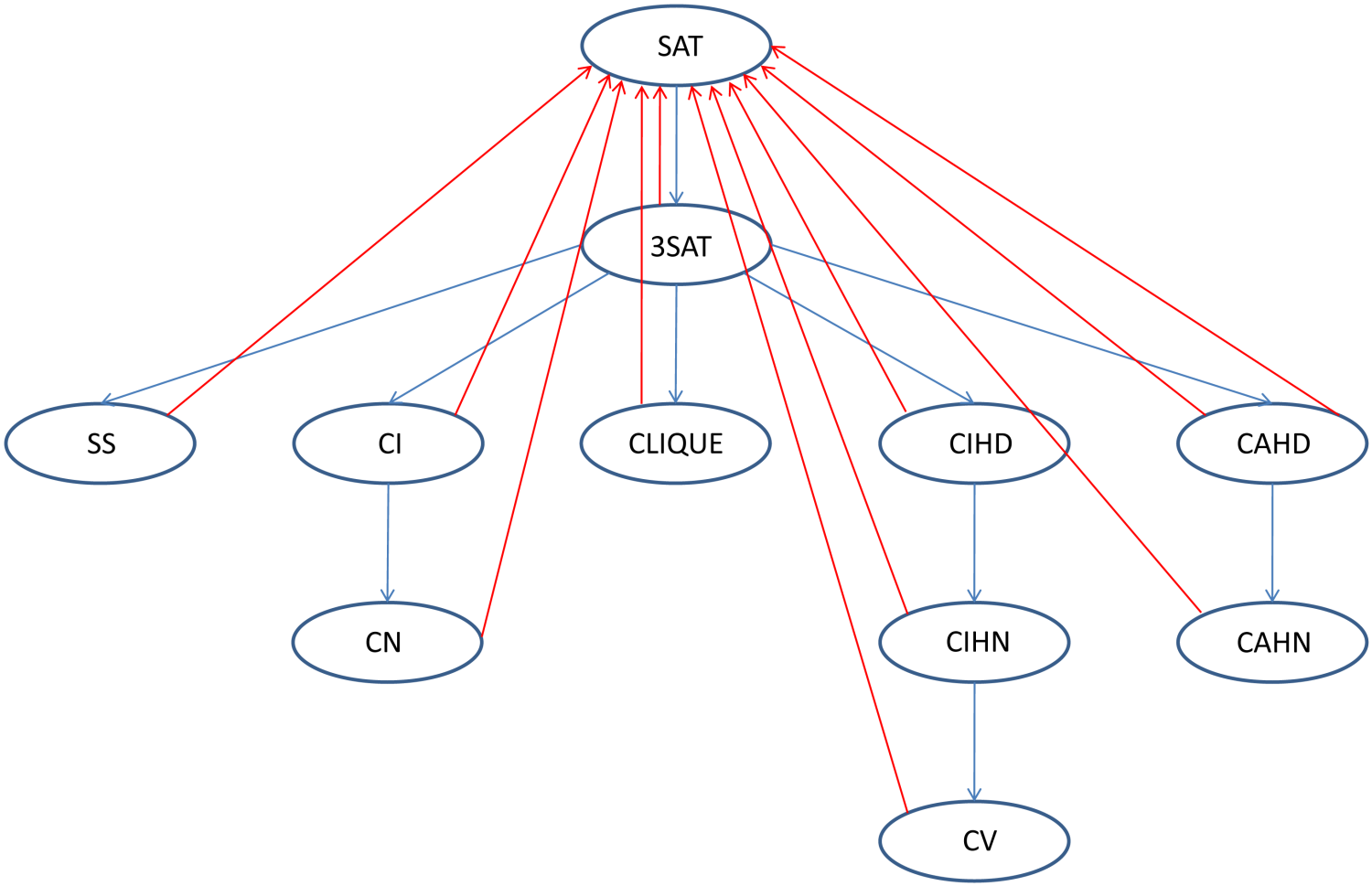
Exemplos de problemas $\mathcal{NP}$-completos

1. *Problema:*
   Soma do subconjunto (SS).
2. *Descrição:*
   Determinar se um conjunto de números inteiros contém algum subconjunto cuja soma dos seus elementos resulte num dado valor t;
3. *Entrada:*
   Um conjunto de valores S e um número t;
4. *Saída:*
   SIM se e somente se S possuir um subconjunto de elementos cuja soma seja t.
5. *Redução:*
   $3SAT$.

Exemplos de problemas $\mathcal{NP}$-completos



Exemplos de problemas $\mathcal{NP}$-completos



Estratégias para lidar com problemas $\mathcal{NP}$-completos

Uma vez que um problema foi identificado como sendo $\mathcal{NP}$-completo, a adoção de uma solução para o caso geral geralmente tem um custo que inviabiliza a sua utilização. Algumas estratégias, no entanto, permitem atenuar tais custos. As principais são:

- Considerar casos particulares;
- Obter soluções aproximadas (para problemas de otimização);
- Efetuar "backtracking" e ramificação limitada;
- Considerar "melhoras locais".

Estratégias para lidar com problemas $\mathcal{NP}$-completos

Casos particulares

Perguntas:

- É realmente necessário considerar o problema em toda a sua generalidade?
- Será que casos particulares (conjuntos restritos de instâncias) não são suficientes para o tipo de aplicação que se pretende?
- Eventualmente a delimitação de casos particulares pode produzir soluções viáveis do ponto de vista prático.

Estratégias para lidar com problemas $\mathcal{NP}$-completos

Casos particulares

Exemplos:

- Não se sabe se SAT e $3SAT$ pertencem à $\mathcal{P}$.
  No entanto, sabe-se que $2SAT \in \mathcal{P}$;
- Vários problemas envolvendo grafos não possuem soluções conhecidas em $\mathcal{P}$.
  No entanto, quando um grafo se reduz a uma árvore (que é um caso particular de um grafo), muitos desses problemas apresentam soluções menos complexas em $\mathcal{P}$.

Estratégias para lidar com problemas $\mathcal{NP}$-completos

Algoritmos de aproximação

Problema de otimização:

- Problema cuja solução possui o melhor "custo" (determinado por uma "função de custo") entre um conjunto de possíveis soluções;
- É diferente de um problema de decisão, pois a resposta não é apenas SIM ou NÃO.

Estratégias para lidar com problemas $\mathcal{NP}$-completos

Algoritmos de aproximação

Exemplo de problema de otimização:

- O *problema de decisão* Caixeiro Viajante determina apenas se existe um Circuito Hamiltoniano com um custo total menor ou igual a um certo valor informado (no caso, a função de custo corresponde à soma dos valores associados aos arcos que formam o circuito);
- O *problema de otimização* Caixeiro Viajante, por outro lado, demanda que se apresente o Circuito Hamiltoniano (se houver) com o menor custo possível entre todos os circuitos que satisfazem o critério do problema de decisão correspondente.

Estratégias para lidar com problemas $\mathcal{NP}$-completos

Algoritmos de aproximação

- Se P é um problema de otimização $\mathcal{NP}$-completo, um "algoritmo de aproximação" para o mesmo pode proporcionar uma solução que não é ótima, mas se aproxima dela de alguma forma;
- Suponha que x seja a entrada do problema e que $opt(x)$ represente a solução ótima;
- Suponha que A é um algoritmo de tempo polinomial para P e que $A(x)$ seja a solução não-ótima produzida por ele para a entrada x;
- Se A satisfaz a desigualdade:

$$\frac{|opt(x) - A(x)|}{opt(x)} \leq \epsilon$$

para algum valor de ϵ, para todas as instâncias x do problema, então A é dito um "algoritmo de ϵ-aproximação".

Estratégias para lidar com problemas $\mathcal{NP}$-completos

Algoritmos de aproximação

Problemas de otimização $\mathcal{NP}$-completos podem ser classificados em:

- Completamente aproximáveis: se, para qualquer valor de $\epsilon > 0$, sempre existir um algoritmo de ϵ-aproximação;
  Pouquíssimos problemas se encaixam nessa categoria (erro arbitrário).
- Parcialmente aproximáveis: se existirem c_1 e c_2 tais que, para qualquer valor $c_1 \leq \epsilon \leq c_2$, sempre existir um algoritmo de ϵ-aproximação;
  Alguns problemas se encaixam nessa categoria (erro limitado).
- Não-aproximáveis: se, para qualquer valor de $\epsilon > 0$, nunca existir um algoritmo de ϵ-aproximação;
  Muitos problemas se encaixam nessa categoria (inclusive o do Caixeiro Viajante).

Estratégias para lidar com problemas $\mathcal{NP}$-completos

"Backtracking" e ramificação limitada

Método baseado nas seguintes etapas:

- Eliminação sistemática de subproblemas do problema original, substituindo-os por subproblemas ainda mais simples mas cuja solução representa a solução do subproblema inicial;
- Verificação se os novos subproblemas gerados no passo acima produzem respostas satisfatórias (as respostas podem ser positivas, negativas ou desconhecidas, no sentido de ser impossível encontrar uma solução rápida para o mesmo).

Essa estratégia produz algoritmos com tempo exponencial no pior caso, mas que freqüentemente apresentam tempos muito melhores do que esse.

Estratégias para lidar com problemas $\mathcal{NP}$-completos

Melhora local

Para problemas de otimização. Método baseado nos seguintes princípios:

- Identificação e utilização de soluções de menor custo localizadas;
- Melhores escolhas dependem do problema e da instância considerada, devendo ser feitas através de experimentação;
- Uso de escolhas aleatórias;
- Exemplos: algoritmos genéticos e redes neurais.

Tempo exponencial no pior caso, não garantem soluções próximas da ótima, mas geralmente apresentam excelentes resultados para problemas $\mathcal{NP}$-completos. Os motivos desse sucesso ainda não são totalmente compreendidos ou antecipados pela teoria da computação.

Leitura complementar

- "The Status of the P Versus NP Problem"
  Lance Fortnow
  Communications of the ACM
  Setembro de 2009

  `http://cacm.acm.org/magazines/2009/9/38904-the-status-of-the-p-versus-np-problem/fulltext`

- "A Teoria da Computação e o Profissional de Informática"
  João José Neto
  RECET
  Outubro de 2009

  `http://revistas.pucsp.br/index.php/ReCET/article/view/1572`